Model Comparison and Assessment by Cross Validation

by

Hui Shen

B.Sc., Computational Mathematics, Nankai University, 1991
M.Sc., Computational Mathematics, Nankai University, 1994
M.Sc., Statistics, National University of Singapore, 2001

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

The Faculty of Graduate Studies

(Statistics)

The University of British Columbia
(Vancouver)

July 2008

Abstract

Cross validation (CV) is widely used for model assessment and comparison. In this thesis, we first review and compare three v-fold CV strategies: best single CV, repeated and averaged CV and double CV. The mean squared errors of the CV strategies in estimating the best predictive performance are illustrated by using simulated and real data examples. The results show that repeated and averaged CV is a good strategy and outperforms the other two CV strategies for finite samples in terms of the mean squared error in estimating prediction accuracy and the probability of choosing an optimal model.

In practice, when we need to compare many models, conducting repeated and averaged CV strategy is not computational feasible. We develop an efficient sequential methodology for model comparison based on CV. It also takes into account the randomness in CV. The number of models is reduced via an adaptive, multiplicity-adjusted sequential algorithm, where poor performers are quickly eliminated. By exploiting matching of individual observations, it is sometimes even possible to establish the statistically significant inferiority of some models with just one execution of CV. This adaptive and computationally efficient methodology is demonstrated on a large cheminformatics data set from PubChem.

Cross validated mean squared error (CVMSE) is widely used to estimate the prediction mean squared error (MSE) of statistical methods. For linear models, we show how CVMSE depends on the number of folds, v, used in cross validation, the number of observations, and the number of model parameters. We establish that the bias of CVMSE in estimating the true

MSE decreases with v and increases with model complexity. In perticular, the bias may be very substantial for models with many parameters relative to the number of observations, even if v is large. These results are used to correct CVMSE for its bias. We compare our proposed bias correction with that of Burman (1989), through simulated and real examples. We also illustrate that our method of correcting for the bias of CVMSE may change the results of model selection.

Table of Contents

List of Tables

List of Figures

Acknowledgements

For the completion of this thesis, firstly, I would like to express my heartfelt gratitude to my supervisor Professor William J. Welch for his invaluable advice and guidance, remarkable insights and continuous encouragement. I have learned many things from him, particularly regarding academic research and analytical writing.

Secondly, I would like to express my sincere appreciation to my former co-supervisor and committee member, Professor Hugh Chipman for the time he dedicated to discuss research work with me, reviewing my thesis and for his precious advice. I would also like to thank my committee member Dr. Matias Salibian-Barrera for his helpful suggestions.

Next, I wish to thank Dr. S. Stanley Young for providing many references in my research area and valuable comments on my research problems. I also wish to thank Professor Jacqueline M. Hughes-Oliver for her remarkable advice on my research paper.

Next, I greatly appreciate my parents for their steadfast support and encouragement.

My special thanks also goes to many individuals:

Professor Jiahua Chen and Bertrand Clark for their insightful discussion with me on my research problem.

Professor Lang Wu and John Petkau for their precious advice on my career.

My department mates Guohua Yan, Wei Wang, Zhong Liu, Juxin Liu, Yiping Dou, Wei Liu, Enqing Shen, Xuekui Zhang for their kind help and support.

My former teammates at University Waterloo, Xu Wang, Yan Yuan, Wanhua Su, Yuanyuan Wang for their support and encouragement.

The faculty, graduate students and staff in the Department of Statistics for providing a supportive and friendly environment and making my stay here an enriching experience.

The research of the thesis was supported by the Natural Sciences and Engineering Research Council of Canada, the National Institutes of Health, USA through the NIH Roadmap for Medical Research, Grant 1 P20 HG003900-01, and the Mathematics of Information Technology and Complex Systems.

Last, but not least, I would like to dedicate this thesis to my husband, Maoxin. He has given me the strongest support through this journey.

Statement of Co-Authorship

The thesis is finished under the supervision of my supervisor, Professor William J. Welch.

Chapter 2 is co-authored with Professors William J. Welch, Hugh Chipman, Jacqueline M. Hughes-Oliver and Dr. S. Stanley Young. My main contributions are performing comparisons of the three Cross-validation strategies, programming for the simulation studies and the real data analysis.

Chapter 3 is co-authored with Professors William J. Welch and Jacqueline M. Hughes-Oliver. Professor Welch and I develop the adaptive CV algorithm and I carry out the programming work for implementing the algorithm.

Chapter 4 is co-authored with Professor William J. Welch. My main contributions are the derivations of the properties of predictive error in linear regression by cross-validation, correction of the estimation for predictive error and programming work in the simulation study and real data analysis.

Chapter 1

Introduction

1.1 Drug Discovery and QSAR Models

The motivation for this thesis comes from drug discovery data. Drug discovery seeks drug candidates by selecting a biological target and screening a large collection of compounds to find compounds active against the target. A target is a body molecule that is thought to play a fundamental role in the onset or progression of a particular disease.

The stages in drug discovery (Törnell and Snaith, 2002) are described as follows: (1) Target identification: identify molecules that clearly play a role in a disease process. (2) Target validation: determine whether intervening at this target will effect an appropriate biological response. (3) Hit identification: a hit is a compound that interacts with a target. (4) Lead generation: once a promising hit has been identified, a diverse combinatorial library is designed to explore the chemical-structure space around that compound and refine its pharmacological profile to produce leads. (5) Lead optimization: additional repeated modifications are made on the leads to produce drug development candidates by optimizing key drug characteristics.

At stage (3), high throughput screening (HTS) is often used, in which very large numbers of compounds are screened against the biological target of interest (Drews, 2000). Although HTS is a highly automated and miniaturized procedure for conducting assays, it is time-consuming, expensive and thus not always feasible for rapidly increasing compound libraries (Valler and Green, 2000, and Service, 1996). Hence sequential screening (Abt, et al., 2001, Engels and Venkatarangan, 2001) has been developed to

help reduce costs and make HTS more efficient.

In sequential screening, the compounds are tested iteratively. Initially, a sample screening set (Young, Lam and Welch, 2002) is tested through HTS and then a quantitative structure activity relationship (QSAR) model is fit using the data from the initial set screened. The QSAR model is then iteratively refined with the new test results. The explanatory variables are so-called chemical descriptors characterizing the chemical structure of a compound. The QSAR model enables prediction of the biological-activity response variable from the chemical descriptors. The latter are easy and cheap to generate by computational chemistry (i.e., no physical measurements are required).

QSAR models are useful for various purposes which include prediction of activities of untested compounds of interest and guiding future screening. QSAR models can reduce cost and time by assaying only those compounds most likely to be active. It is even possible to carry out virtual screening: predict the activity of compounds that have never been made yet and are not even part of a chemical library. For reviews of QSAR models, see Free and Wilson (1964), Hansch and Fujita (1964), Lipnick (1986), and Kubinyi (2002).

Finding effective QSAR models is challenging due to the following reasons:

(1) For drug discovery data, active compounds are usually rare, thus data are imbalanced.

(2) Drug discovery data sets are usually very large in both the number of compounds and descriptors. In some situations, the number of descriptors can exceed the number of compounds.

(3) The relationships between some descriptor sets and activity are highly non-linear.

1.2 Overview of Some Modeling Methods in the Thesis

QSAR models used in the thesis include neural networks (NN), random forests (RF) and k-nearest neighbors (KNN). I will give a brief introduction of these methods although my thesis is focused on assessment of them instead of developing a new method or improving current methods.

1.2.1 Neural networks (NNs)

NNs (Ripley, 1996) are frequently used in classification problems to model conditional probabilities of class membership given observed features (explanatory variables). Given enough training data, in principle a neural network can data-adaptively model a complex relationship between the class response and the explanatory variables. A feed-forward NN with one hidden layer is the simplest but most common form (Ripley, 1996).

Figure 1.1 (Welch, 2002) shows a neural network for a two-class classification problem with three input (explanatory) variables and a single output (response) variable. Between the input and output layers is a hidden layer. The hidden layer can have any number of units; here it has just two, labeled H1 and H2. Inputs feed forward through the network. In Figure 1.1, for the hidden layers units, the inputs are x_1, x_2 and x_3, the actual input variables and the constant input $x_0 = 1$. The units take weighted sum of their inputs and apply transfer functions like the logistic transformation to generate the unit's output. Similarly, a transfer function is applied on weighted sum of the hidden units. Let d be the number of input variables, and h be the number of hidden units. The number of weights is $(d+1)\times h+h+1 = (d+2)\times h+1$.

Figure 1.2 shows a neural network with three input units, three hidden units and a single output. We can see that the number of weights will increase by $d + 2 = 3 + 2 = 5$ compared to Figure 1.1. If d or h is very big, the number of weights can be very large and fitting the neural network

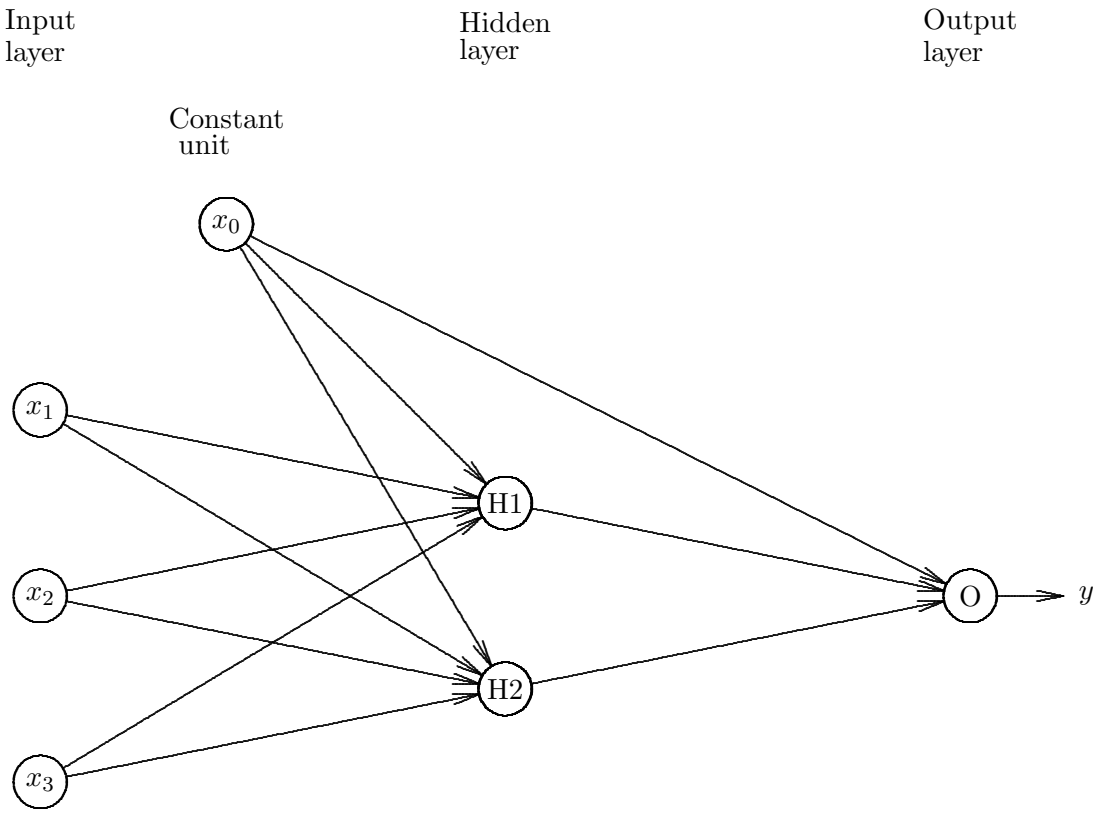

Figure 1.1: Neural network with $d = 3$ input units, two hidden units, H1 and H2, in a single layer, and a single output unit, O.

would be very computationally demanding.

In NN, the user is spared the burden of specifying the functional form of the input-output relationship. However, the user has to specify a network architecture and other parameters. Among all the parameters, "size" and "decay" are the most important to influence the model fitting, where "size" is the number of units in the hidden layer and "decay" is a penalty for the sum of squares of the weights. It shrinks the weights and makes the model less complex. Hence, both size and decay affect the model complexity. We will use these two tuning parameters for the NN method in Chapter 3 and 4.

1.2.2 Random forest (RF)

A random forest, developed by Breiman (2001), is an ensemble of many classification trees built from random sub samples (bootstrap samples) of the training data, and using random subsets of the available variables during the building of each classification tree. Together the classification trees of the forest represent a model where each decision tree votes for classifying a new test object, and the majority wins the vote.

A random forest is typically made up of tens or hundreds of trees. Each tree is built from an independent bootstrap sample of the training data. Also, in building each tree, a different subset of the available variables is used at each and every node to choose the variable to partition the data. With so much randomness, substantial performance gains and considerable robustness to noise, outliers, and over-fitting is achieved when compared to a single tree classifier.

1.2.3 K-nearest neighbors (KNN)

KNN is a very simple but effective non-parametric method (see Dasarathy, 1990 for a review). It estimates the class probability or response value of a new case according to its k neighbors in the explanatory variable space.

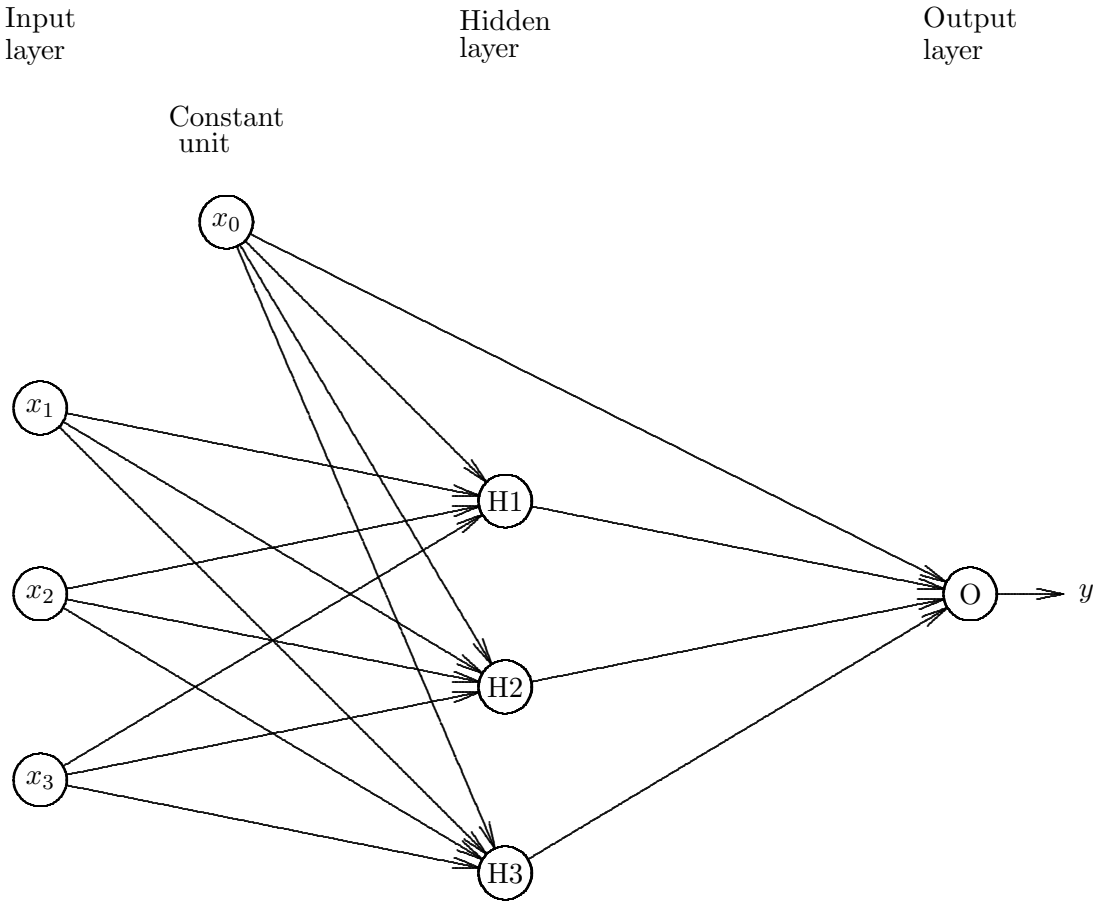

Figure 1.2: Neural network with $d = 3$ input units, three hidden units, H1, H2 and H3, in a single layer, and a single output unit, O.

The algorithm is as follows: for each observation in the test set, the k nearest points in the training set based on some proximity measure are found. The prediction is then made by either the majority vote among the selected closest points (classification) or the mean response of the selected closest points (regression). Euclidean distance is a commonly used proximity measure. The parameter k, number of the neighbors, is chosen by the user, often by cross validation.

1.3 Cross Validation

When QSAR models are fit, their predictive performance needs to be assessed. Cross Validation (CV) is a method widely used in predictive performance estimation and model comparison. CV was invented by Stone (1959) in linear regression where he wanted to balance between the need to fit and the need to assess a model.

There exist many versions of CV and the most commonly used is v-fold CV. In this procedure, the data set is randomly split into v parts or folds each containing approximately equal number of data points. The first part is left out for testing , and the remaining $v-1$ parts form the training data. A given model is fit on the training data and then applied to the test set to obtain predictions. The process is repeated, leaving out each of the v parts in turn. In this way all the data are used for training and for testing, but the assessment is honest as we never use a case simultaneously in the training and the test sets. An illustration is given in Table 1.1 for a 10-fold CV procedure.

We will introduce various types of CV in detail in Chapter 2. Different performance measures are used in CV for different types of response. For a continuous response, mean squared prediction error is popular. For a binary response, researchers often use misclassification rate. For drug discovery data, misclassification rate is not an appropriate performance measure due to the active class being rare. Instead, measures based on prioritizing or

Table 1.1: 10-fold CV

Test data	Training data		Get prediction
Fold 1	Fold 2, 3,..., 10	$\Rightarrow$	for all cases in fold 1
Fold 2	Fold 1, 3,..., 10	$\Rightarrow$	for all cases in fold 2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
Fold 10	Fold 1, 2,..., 9	$\Rightarrow$	for all cases in fold 10

ranking test compounds in terms of their probabilities of activity are preferred. A specific summary of the effectiveness of the ranking, AHR (average hit rate), will be described in Chapter 2. Alternatively, the number of hits found when a certain number of compounds is selected will be introduced in Chapter 3 and the mean squared prediction error will be used in Chapter 4 for regression problems.

To date, much research work has been done on CV as described in Chapter 2, 3 and 4. Some asymptotic results of CV have been derived in estimating predictive performance and model selection. However, these asymptotic results may not be applicable to finite samples, or to drug discovery data because of their modelling challenges. In the thesis, we will focus on the properties of CV in performance prediction and model selection for finite samples. Computationally efficient algorithm will also be developed.

1.4 Brief Introduction of the Thesis

The thesis is "manuscript-based", where each chapter is self-contained, including references. Properties of CV and efficient algorithms are a unifying theme, however.

The thesis is organized as follows. In Chapter 2, we review and compare three v-fold CV strategies: best single CV, repeated and averaged CV and double CV. In best single CV, v-fold CV is carried out for one random split of the data. In repeated and averaged CV, v-fold CV is implemented

for S random splits of the data and predictive performance is averaged over S splits. In double v-fold CV, a two-stage v-fold CV is conducted, separating model selection via CV from model assessment, for one random split of the data. The predictive performances of the three CV strategies in terms of the AHR criterion are illustrated by using simulated and real data examples from PubChem (`http://pubchem.ncbi.nlm.nih.gov`). Chapter 2 shows that repeated-and-averaged CV is superior, though computationally demanding.

In practice, when we need to compare many models, conducting repeated and averaged CV strategy is not computational feasible. In Chapter 3, we develop an efficient sequential algorithm based on CV for model comparisons. It also takes into account the randomness in CV. Nonetheless, by exploiting matching of observations across the CV analyses for several modelling strategies, it is sometimes possible to establish the statistically significant superiority of one strategy versus another with just one execution of CV. Thus, many possible modelling strategies are quickly eliminated as inferior. This adaptive and computationally efficient methodology is demonstrated on a large cheminformatics data set from PubChem (`http://pubchem.ncbi.nlm.nih.gov`).

Both Chapter 2 and Chapter 3 use drug discovery data sets with binary response. The performance measures used are AHR and the number of hits at 300 compounds selected, which are appropriate for drug discovery data. These two performance measures are based on ranking, making theoretical analysis difficult.

Although linear regression is not an appropriate model for drug discovery data, we hope that investigating the theoretical properties of CV based on this simplest model for finite samples provides some useful insight for QSAR models in general. In Chapter 4, we study the properties of CV in terms of estimating mean squared prediction error (CVMSE) in linear regression. We compare the estimation properties for different numbers of folds, v, in

CV and derive the bias of CVMSE. We also propose a correction method to reduce the bias. This chapter helps to explain some of the large biases found empirically in Chapter 2.

Chapter 5 gives a summary of the thesis and provides some directions for future research.

Bibliography

[1] Abt, M. J., Lim, Y.-B., Sacks, J., Xie, M. and Young, S. S. (2001). A Sequenal Approach for Identifying Lead Compounds in Large Chemical Databases, *Stat. Sci.* 16, 154–168.

[2] Breiman, L. (2001). Random Forest, *Machine Learning* 45(1), 5–32.

[3] Dasarathy, B. V. (1990). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, Los Alamitos, Calif.: IEEE Computer Society Press.

[4] Drews, J. (2000). Drug Discovery: A historical Perspective, *Science* 287, 1960–1965.

[5] Engels, M. F. M., and Venkatarangan, P. (2001). Approaches to Efficient HTS, *Curr. Opin. Drug Discovery Dev.* 4, 275–283.

[6] Free, S. M., and Wilson, J. W. (1964). A Mathematical Contribution to Structure-Activity Studies, *Journal of Medicinal Chemistry* 7, 395–399.

[7] Hansch, C., and Fujita, T. (1964). r-s-p Analysis - A Method for the Correlation of Biological Activity and Chemical Structure, *Journal of the American Chemical Society* 86, 1616–1626.

[8] Kubinyi, H. (2002). From Narcosis to Hyperspace: The History of QSAR, *Quantitative Structure-Activity Relationships* 21(4), 348–356.

[9] Lipnick, R. L. (1986). Charles Ernest Overton: Narcosis Studies and a Contribution to General Pharmacology, *Trends in Pharmacological Sciences* 7(5), 161–164.

[10] Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.

[11] Service, R. F. (1996). Combinational Chemistry Hits the Drug Market, *Science* 272, 1266–1268.

[12] Stone, M. (1959). Application of a Measure of Information to the Design and Comparison of Regression Experiments, *Annals Math. Stat.* 30 55–70.

[13] Törnell, J. and Snaith, M. (2002). Transgenic Systems in Drug Discovery: from Target Identification to Humanized Mice, *Drug Discovery Today* Volume 7, Issue 8, 461–470.

[14] Valler, M. J., and Green, D. (2000). Diversity Screening Versus Focussed Screening in Drug Discovery, *Drug Discovery Today* 5, 286–293.

[15] Welch, W. J. (2002). *Computational Exploration of Data*, Stat441/841 Course Notes, University of Waterloo.

[16] Young, S. S, Lam, R. and Welch, W. J. (2002). Initial Compound Selection for Sequential Screening, *Current Opinion in Drug Discovery and Development*, 5(3): 422–427.

Chapter 2

Cross Validation Strategies for Model Assessment and Selection

2.1 Introduction

Cross validation (CV) is widely used for model assessment and selection. In this article, we consider three CV strategies for tuning and selecting neural networks or random-forest models, though the approach would apply more generally. Some commonly used CV strategies include "hold out set" CV, v-fold CV and leave-one-out CV.

In "hold out set" CV, the data are randomly split into two parts: one part is called the training data and the other is called the test data. The CV procedure is to fit a model on the training data and then use the fitted model to predict the response values in the test data.

In v-fold CV, the data are randomly split into v groups or folds, one group is considered as test data for assessing prediction accuracy, and the other v-1 groups are considered as training data and used for model fitting. This process is repeated with each of the groups in turn as test data. Leave-one-out CV is a special case of v-fold CV when v equals the number of

A version of this chapter will be submitted for publication. Authors: Shen H., Welch W. J., Chipman H. A., Hughes-Oliver J. M., Young S. S.

observations in the data, i.e., the test data have only one observation.

Much theoretical work has been done on CV. Stone (1974, 1977) focused mainly on properties for leave-one-out CV. Li (1987), Shao (1993) and Zhang (1993) investigated CV procedures for linear models. Burman (1989) established theoretical results for v-fold CV and general models. More recently, Dudoit and van der Laan (2003) derived asymptotic properties of a broad definition of CV (e.g., leave-one-out, v-fold, Monte Carlo, etc.) for model selection and performance assessment. Efron (2004) explored the connection between cross-validation and some penalty methods in terms of prediction error. So far, all the theoretical results on CV are asymptotic and for large samples. For finite samples, the properties of CV could be very complicated and unclear. In this paper, we will focus on comparing different v-fold CV strategies for finite samples.

In practice, more and more researchers have been using CV for selecting and assessing models, e.g., Dietterich (1998), Hawkins et al. (2003), Sinisi and van der Laan (2004), Hughes-Oliver et al. (2007).

CV strategies are usually aimed at assessing (estimating) predictive performance, and different performance measures are used for different types of response. For a continuous response, mean squared prediction error is a popular measure. For a binary response, researchers often use misclassification rate. In this paper, we only consider data with binary response and will use specialized performance measures appropriate for the objectives of modelling drug-discovery data (Section 2.2) .

Suppose we have a given data set with binary response y and chemical-descriptor variables X. Given l candidate models, $m_1, \ldots, m_l$, our goal is to use a good v-fold CV strategy to estimate their predictive performances accurately and select an optimal model with the best performance.

The paper is organized as follows. Section 2.2 gives the definition of the performance measure we will use, average hit rate (AHR). Section 2.3 introduces three v-fold CV strategies for estimating performance. In Section

2.4, we compare the three CV strategies using a simulated example and Section 2.5 makes similar comparisons for a real drug-discovery data set. Conclusions and discussions are presented in Section 2.6.

2.2 Predictive Performance Measure

2.2.1 Average hit rate

In this paper, we will use average hit rate (AHR) as the performance measure. AHR is discussed in detail in Wang (2005). It is a popular criterion for unbalanced binary data where the class of interest is rare. With drug discovery data, active compounds are very rare and we want to find them with minimal screening. In this situation, usual performance measures such as misclassification rate are not effective.

Assume there are n compounds in a data set, the assay response is binary (active/inactive), and active compounds are of interest. Suppose that some model assigns an estimated probability of activity, $\hat{p}$, to each compound (possibly via cross validation), and their estimated probabilities are sorted in descending order. Denote the response values for the ordered list of compounds by $y_{(1)}, y_{(2)}, \ldots, y_{(n)}$. Then AHR is defined as

$$\text{AHR} = \frac{\sum_{i=1}^{n} (\frac{y_{(i)} \sum_{j=1}^{i} y_{(j)}}{i})}{A},$$

where A is the number of active compounds in the data, and

$$y_{(i)} = \begin{cases} 1 & \text{if the } (i)\text{th compound is active} \\ 0 & \text{otherwise.} \end{cases}$$

Now we look at an example. For a hypothetical data set, the observed responses are 10 values with

$$y = (0, 1, 0, 0, 0, 1, 1, 0, 1, 0),$$

Suppose that based on a certain model, we get predicted $\hat{p}$ values as follows

$$\hat{p} = (0.7, 0.4, 0.6, 0.5, 0.3, 0.2, 0.8, 0.1, 0.9, 0),$$

and thus the sorted $\hat{p}$ values in descending order are

$$\hat{p}_{\text{sort}} = (0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0),$$

with corresponding y

$$y_{\text{sort}} = (1, 1, 0, 0, 0, 1, 0, 1, 0, 0).$$

If compounds are selected in the order given by $\hat{p}_{\text{sort}}$, 1 active compound is found at the first selection, and there are 2, 2, 2, 2, 3, 3, 4, 4, 4 active compounds found in the first 2-10 compounds selected, respectively. Thus according to the definition of AHR,

$$\text{AHR} = \frac{1}{4}\left(1\times\frac{1}{1}+1\times\frac{2}{2}+0\times\frac{2}{3}+0\times\frac{2}{4}+0\times\frac{2}{5}+1\times\frac{3}{6}+0\times\frac{3}{7}+1\times\frac{4}{8}+0\times\frac{4}{9}+0\times\frac{4}{10}\right) = \frac{3}{4}.$$

It can be seen that AHR is a value between 0 and 1. AHR gives more weight to early selections; thus it describes how quickly active compounds are found as well as how many. A higher value of AHR indicates a better ranking scheme.

If there are tied $\hat{p}$ values, AHR depends on the random ordering of compounds with equal $\hat{p}$ values. Expectation can be taken with respect to all the possible orderings with ties. Wang (2005) provided an algorithm to compute expected AHR in such situations.

2.2.2 True performance

Let T be the true predictive performance based on a given model and data set (y, X). For example, T could be the AHR measure described above. Ideally, we want to assess performance for compounds drawn from a bigger

population ("new compounds"). We will define T as the expectation of the performance measure w.r.t. the many possible test sets.

Suppose there are l models, denoted by $m_1, \ldots, m_l$. Let T_i be the true predictive performance for m_i, where $i = 1, \ldots, l$. Let T_{opt} be the optimal value of T, i.e., $T_{\text{opt}} = \max\{T_1, \ldots, T_l\}$, and let m_{opt} be the optimal model, i.e., the one giving T_{opt}. In practice, T_i and T_{opt} are unknown. Note that all the candidates models may be wrong in the sense that none generated the data, thus T_{opt} is the optimal T only among the l candidate models.

2.3 Cross Validation

2.3.1 Three cross validation strategies

Our goals are to estimate the predictive performance of each candidate model accurately in terms of T and select an optimal model $\hat{m}_{\text{opt}}$ among the l candidate models by a CV strategy. For model selection, we hope that $\hat{m}_{\text{opt}}$ is actually the true optimal model, m_{opt}, or from equivalently good models.

Denote by $\hat{T}$ the estimate of T from a CV strategy. In this paper we consider conducting three v-fold CV strategies.

(1) Single v-fold CV. In this strategy, v-fold CV is carried out for each model for *one* random split of the data. The split is the same for each model. $\hat{T}$ (based on AHR, say) is computed for each model and then the maximum $\hat{T}$ is an estimate of T_{opt}. $\hat{m}_{\text{opt}}$ is the model with the the maximum $\hat{T}$.

(2) Repeated and averaged v-fold CV. Burman (1989) developed this method and established some theoretical results for general models and a continuous response. In this strategy, v-fold CV is carried out for s random splits of the data and $\hat{T}$ is computed and averaged over the s splits for each model. Then the maximum average $\hat{T}$ is an estimate of T_{opt}. $\hat{m}_{\text{opt}}$ is the model with the maximum average $\hat{T}$. This strategy attempts to reduce,

by averaging, the randomness inherent in randomly splitting the data into groups ($v < n$).

(3) Double v-fold CV. Double CV was first developed by Stone (1974) and has been widely applied since then, e.g., Mosteller and Tukey (1977), and Hawkins et al. (2003). In this strategy, for each model, a two-stage v-fold CV is carried out for one random split of the data.

Taking $v = 4$ for illustration, in double 4-fold CV, at the first stage the whole data are divided into 4 parts, named A, B, C and D. The training/test data partitions are BCD/A, ACD/B, ABD/C and ABC/D. For each training data set, e.g., BCD, we conduct a 3-fold CV, i.e., each candidate model is fit on BC, BD and CD and predictions made on D, C and B, respectively. Then we select an "optimal" model m_A based on these D, C and B predictions. Similarly, for the training data sets ACD, ABD and ABC, we can conduct 3-fold CV to get optimal models, namely m_B, m_C and m_D. At the second stage, the optimal model m_A is refit on BCD to predict A, m_B is refit on ACD to predict B, m_C is refit on ABD to predict C, and m_D is refit on ABC to predict D. Combining the predictive $\hat{p}$ values for the four parts we get the overall $\hat{T}$ for the entire data set.

For any CV strategy, let $\hat{T}_{ij}$ be $\hat{T}$ for model m_i and split j of s splits of the data ($s \geq 1$ only for repeated and averaged v-fold CV). Since splits are random, conditional on i, the $\hat{T}_{ij}$ are independent for $j = 1, \ldots, s$. For single CV, only 1 split is performed, so we observe just one realization, $\hat{T}_{i_1}$, for m_i. Thus, $\hat{m}_{\text{opt}} = \text{argmax}\{\hat{T}_{i_1}\}, i = 1, \ldots, l$, and $\hat{T}_{\text{sing}}$ is max $\hat{T}_{i_1}$. For repeated and averaged CV, $\hat{m}_{\text{opt}} = \text{argmax}\{\bar{\hat{T}}_{i.}\}$, where $\bar{\hat{T}}_{i.} = \dfrac{1}{s}\sum_{j=1}^{s} \hat{T}_{ij}$, and $\hat{T}_{\text{rep}} = \max_i\{\bar{\hat{T}}_{i.}\}, i = 1, \ldots, l$. For double CV, $\hat{T}_{\text{doub}}$ is defined like single CV, but different "best" models may be chosen by model selection at the first stage of CV, and there may be no unique $\hat{m}_{opt}$.

2.3.2 Computational complexity

For single v-fold CV, each model is fit v times and predicted v times. Since model fitting typically takes much longer time than predicting, we will ignore the computer time for predicting. Suppose one model fit takes 1 unit of computer time, then for l models, the computing time is vl units.

For repeated and averaged CV, if we implement the CV for s splits of the data, the computing time is svl units.

For double CV, the computing time for the first stage and the second stage is $v(v-1)l$ and vl units respectively. But by reusing fits, we can save half computer time for the first stage. For the double 4-fold CV example in Section 2.3.1, at the first stage, suppose the training/test data partitions are BCD/A, ACD/B, ABD/C and ABC/D. For training data BCD, we fit models using BC, BD and CD. Similarly, for the training data ACD, we fit models using AC, AD and CD. Note that CD can be fit once and used twice for prediction, similarly for other data partitions. Thus, the number of model fits is reduced from $12l$ to $6l$. At the second stage, v models are refit in general. Therefore, the total computer time is $(\frac{v(v-1)}{2}+v)l = \frac{v(v+1)l}{2}$ units.

2.3.3 Performance of CV in assessing predictive accuracy

In this section, we will assess the predictive performance of the three CV strategies based on two criteria: 1) MSE, the mean squared error of $\hat{T}$ in estimating T_{opt}, or equivalently RMSE, the root mean squared error; and 2) the probability of choosing the correct optimal model, $Pr(m_{\text{opt}}$ is chosen). If there are some models which are very close to each other in RMSE values, we will treat all these models as optimal models.

Let MSE_{sing}, MSE_{rep} and MSE_{doub} be the MSE for single, repeated and averaged, and double v-fold CV respectively. Then we have

$$\text{MSE}_{\text{sing}} = (E(\hat{T}_{\text{sing}}) - T_{\text{opt}})^2 + \text{Var}(\hat{T}_{\text{sing}}), \tag{2.1}$$

$$\text{MSE}_{\text{rep}} = (E(\hat{T}_{\text{rep}}) - T_{\text{opt}})^2 + \text{Var}(\hat{T}_{\text{rep}}), \tag{2.2}$$

and

$$\text{MSE}_{\text{doub}} = (E(\hat{T}_{\text{doub}}) - T_{\text{opt}})^2 + \text{Var}(\hat{T}_{\text{doub}}), \tag{2.3}$$

where $E(\hat{T}_{\text{sing}}) - T_{\text{opt}}$, $E(\hat{T}_{\text{rep}}) - T_{\text{opt}}$ and $E(\hat{T}_{\text{doub}}) - T_{\text{opt}}$ are the biases for the three CV strategies. Let RMSE be the root mean squared error, i.e., $\text{RMSE}_{\text{sing}} = \sqrt{\text{MSE}_{\text{sing}}}$, etc.

Recall that T_{opt} is the conditional expectation of predictive performance for the optimal model, thus the expectations in formulas (2.1), (2.2) and (2.3) are conditioning on the given data as well and with respect to random splits of the data.

Besides computing T_{opt}, we are also interested in estimating $P_r(m_{\text{opt}}$ is chosen) for the three CV strategies.

Theoretically, it is hard to get an explicit expression for T_{opt}, $P_r(m_{\text{opt}}$ is chosen), $E(\hat{T}_{\text{sing}})$, $\text{Var}(\hat{T}_{\text{sing}})$ etc., thus we have to estimate them approximately using numerical methods. For real data, it is usually impossible even to estimate T_{opt} numerically. However, we are able to do so using simulated data.

Suppose we have generated simulated training data with sample size n, binary response y and chemical-descriptor (explanatory) variables X. To estimate T_{opt}, we need to estimate T_i for model m_i, where $i = 1, \ldots, l$. T_i can be estimated by Monte Carlo simulation as follows. Let $\tilde{T}_i$ be a Monte Carlo estimate of T_i, $i = 1, \ldots, l$. First, we fit model m_i using all the training data (y, X), then we generate test X and test y based on the same data-generating mechanism as used for the training data, and compute an estimate of T_i. The final Monte Carlo estimate, $\tilde{T}_i$ is the average of those estimates over many test sets. We will use this $\tilde{T}_i$ as an estimate of T_i as long as the standard error of $\tilde{T}_i$ (i.e., the standard error of a mean) is very small. From now on, we will not distinguish T and $\tilde{T}$. Similarly,

$$\tilde{T}_{\text{opt}} = \max_i \{\tilde{T}_i\}, i = 1, \ldots, l, \tag{2.4}$$

and we will simply plug $\tilde{T}_{\text{opt}}$ in for T_{opt} in (2.1), (2.2) and (2.3), ignoring Monte Carlo error.

In this paper we use 10-fold CV for the three CV strategies. For single CV, to estimate $E(\hat{T}_{\text{sing}})$ and $\text{Var}(\hat{T}_{\text{sing}})$, we compute $\hat{T}_i$ by 10-fold CV based on s random splits of the data for each model. In practice, by definition, single CV is based on just one split. However, to estimate true frequentist performance of single CV, we consider many hypothetical repeats of the process to estimate the first two moments of $\hat{T}_{\text{sing}}$. Thus we have

$$\hat{E}(\hat{T}_{\text{sing}}) = \frac{1}{s}\sum_{j=1}^{s} \max_{i=1,\ldots,l}\{\hat{T}_{ij}\}, \tag{2.5}$$

$$\widehat{\text{Var}}(\hat{T}_{\text{sing}}) = \frac{1}{s-1}\sum_{j=1}^{s}(\max_{i=1,\ldots,l}\{\hat{T}_{ij}\} - \hat{E}(\hat{T}_{\text{sing}}))^2. \tag{2.6}$$

For the repeated and averaged CV strategy, suppose $\hat{T}_{\text{rep}}$ is obtained for model m_{i_0}. The estimates of $E(\hat{T}_{\text{rep}})$ and $\text{Var}(\hat{T}_{\text{rep}})$ should be obtained from many repeats of the repeated and averaged CV procedure. However, as long as the standard error of $\hat{T}_i$ is small for all $i = 1, \ldots, l$, we can estimate $E(\hat{T}_{\text{rep}})$ by $\hat{T}_{\text{rep}}$, one realization of repeated and averaged CV implementation. Thus,

$$\hat{E}(\hat{T}_{\text{rep}}) = \hat{T}_{\text{rep}}, \tag{2.7}$$

$$\widehat{\text{Var}}(\hat{T}_{\text{rep}}) = \frac{1}{s}\widehat{\text{Var}}(\hat{T}_{i_0 j}), j = 1, \ldots, s. \tag{2.8}$$

For double CV, by a similar argument to that for single CV, $E(\hat{T}_{\text{doub}})$ and $\text{Var}(\hat{T}_{\text{doub}})$ are estimated by the sample average and variance of $\hat{T}$ over s random splits of the data.

For each of the three CV strategies, $P_r(m_{\text{opt}}$ is chosen), the probability of choosing the optimal model, is estimated by the proportion of choosing the correct model, m_{opt}, among the total number of 10-fold CV executions.

Next, we will use a simulation example to compare the three CV strate-

gies.

2.4 Simulated Example

2.4.1 Data generation

The simulated data are generated once according to the description at `www.stat.ncsu.edu/research/drug_discovery/SimulatedDataset/`, the website of the North Carolina State University Exploratory Center for Cheminformatics Research, with a minor change. The response variable y is binary, taking values 0 (inactive) and 1 (active), and the training sample size is 1000. There are 100 binary chemical-descriptor variables in X. The training set y is generated such that

$$P_r(y_i = 1) = 0.9, \text{ for } i = 1, \ldots, 80; \quad P_r(y_i = 1) = 0.01, \text{ for } i = 81, \ldots, 1000.$$

That is, the first 80 compounds in the sample are active and the rest are inactive, both with a little noise. The active compounds arise from the following 4 mechanisms.

Mechanism 1: Compounds $1 - 20$ have variables 1, 2, 3, 4, and 5 taking values 1, 0, 1, 0, 1, respectively.

Mechanism 2: Compounds $21 - 40$ have variables 5, 6, 7, 8 and 9 taking values 0, 1, 1, 1, 1, respectively.

Mechanism 3: Compounds $41 - 60$ have variables 3, 10, 11, 12 and 15 taking values 1, 1, 1, 1, 1, respectively.

Mechanism 4: Compounds $61 - 80$ have variables 13, 14, 15, 16 and 17 taking values 1, 1, 0, 1, 1, respectively.

The remaining X variables are generated from independent Bernoulli(0.01) distributions.

2.4.2 Results based on neural network (NN)

In this section we assess the neural network (NN) method for the three CV strategies and the simulated data . NN (Ripley, 1996) is frequently used in classification problems to model conditional probabilities of class membership given observed features (chemical-descriptor variables). Given enough training data, in principle a neural network can approximate any smooth functional relationship for the true probability of being active. Furthermore, the user is spared the burden of specifying the functional form. However, the user has to specify a network architecture and other parameters. Among all the parameters, "decay" and "size" are the most important to influence the model fitting, where "size" is the number of units in the hidden layer, and "decay" is a penalty for the sum of squares of the fitted weights. Thus we will compare different NN models, with different size and decay values. The models used are summarized in Table 2.1.

Table 2.1: NN models for the simulated data

	Model		
size	decay=0.1	decay=0.01	decay=0.001
4	m_1	m_2	m_3
5	m_4	m_5	m_6
7	m_7	m_8	m_9
9	m_{10}	m_{11}	m_{12}

For each model, Table 2.2 displays the average value of $\hat{T}$ over 100 splits for repeated and averaged 10-fold CV and T. The standard error of the average $\hat{T}$ and the Monte Carlo standard error in estimating T are given in brackets. Note that the values are AHR (larger-the-better). We can see from Table 2.2 that $T_{\text{opt}} = 0.324$, which is obtained at m_{10}.

$\hat{T}_{\text{rep}} = 0.301$ with standard error 0.0026 which is very small. Therefore

Table 2.2: True performance, T, and estimated performance, $\hat{T}$, from 10-fold CV averaged over 100 splits, for NN models and the simulated data. The standard error of the average $\hat{T}$ and the Monte Carlo standard error in estimating T are given in brackets.

Estimated/True	size	decay=0.1	decay=0.01	decay=0.001
$\hat{T}$	4	0.228 (0.0026)	0.195 (0.0025)	0.173 (0.0024)
T		0.249 (0.0008)	0.228 (0.0006)	0.250 (0.0006)
$\hat{T}$	5	0.247 (0.0031)	0.214 (0.0032)	0.204 (0.0027)
T		0.282 (0.0008)	0.265 (0.0008)	0.191 (0.0006)
$\hat{T}$	7	0.284 (0.0032)	0.255 (0.0027)	0.238 (0.0033)
T		0.309 (0.0009)	0.284 (0.0008)	0.286 (0.0009)
$\hat{T}$	9	0.301 (0.0026)	0.275 (0.0034)	0.263 (0.0033)
T		0.324 (0.0010)	0.290 (0.0009)	0.305 (0.0008)

we can use (2.7) and (2.8) and get

$$\hat{E}(\hat{T}_{\text{rep}}) = 0.301, \quad \widehat{\text{Var}}(\hat{T}_{\text{rep}}) = (0.0026)^2.$$

Thus,

$$\widehat{\text{MSE}}_{\text{rep}} = (0.324 - 0.301)^2 + 0.0026^2 = 0.000536, \quad \widehat{\text{RMSE}}_{\text{rep}} = 0.0231.$$

For the single 10-fold CV strategy, we estimate $E(\hat{T}_{\text{sing}})$ and $\text{Var}(\hat{T}_{\text{sing}})$ using (2.5) and (2.6) based on 100 random splits of the data and get

$$\hat{E}(\hat{T}_{\text{sing}}) = 0.317, \quad \widehat{\text{Var}}(\hat{T}_{\text{sing}}) = (0.0255)^2.$$

Therefore,

$$\widehat{\text{MSE}}_{\text{sing}} = (0.324 - 0.317)^2 + 0.0255^2 = 0.000699, \quad \widehat{\text{RMSE}}_{\text{sing}} = 0.0264.$$

Column 2 of Table 2.3 shows the results for single CV for 10 (of the 100)

splits of the data. The model number in brackets is the model chosen for each split.

Table 2.3: Optimal $\hat{T}$ for NN and RF models and the simulated data, from single and double 10-fold CV, for 10 splits of the data

	$\hat{T}$			
	NN		RF	
Split	Single	Double	Single	Double
1	0.320 (m_7)	0.269	0.439 (m_{14})	0.469
2	0.314 (m_8)	0.256	0.444 (m_{13})	0.429
3	0.304(m_8)	0.276	0.464 (m_{13})	0.461
4	0.314 (m_{10})	0.277	0.434 (m_{13})	0.414
5	0.324 (m_{10})	0.262	0.477 (m_{13})	0.459
6	0.324(m_{10})	0.244	0.425 (m_{13})	0.426
7	0.300 (m_{10})	0.316	0.444 (m_{10})	0.434
8	0.303 (m_{10})	0.305	0.476 (m_{13})	0.466
9	0.324 (m_{10})	0.281	0.439 (m_{13})	0.405
10	0.309 (m_{10})	0.319	0.441 (m_{13})	0.448

We run double 10-fold CV for only 10 random splits of the data since the computer time is comparable to that for repeated and averaged 10-fold CV based on 100 random splits of the data. The results are shown in column 3 of Table 2.3 , where the 10 splits are the same for double and single 10-fold CV. From Table 2.3 , we get

$$\hat{E}(\hat{T}_{\text{doub}}) = 0.281, \quad \widehat{\text{Var}}(\hat{T}_{\text{doub}}) = (0.0253)^2.$$

Thus,

$$\widehat{\text{MSE}}_{\text{doub}} = (0.324 - 0.281)^2 + 0.0253^2 = 0.00249, \quad \widehat{\text{RMSE}}_{\text{doub}} = 0.0499.$$

Taking a further look at columns 2 and 3 of Table 2.3 , we see that for splits 2 and 3, single CV chooses m_8 as the optimal model which is actually

a sub-optimal model from Table 2.2. However, the values of $\hat{T}_{\text{sing}}$, 0.314 and 0.304, for the two splits are much bigger than the average of $\hat{T}$ for m_8, i.e., 0.255. Another interesting finding is that for split 6, $\hat{T}_{\text{sing}} = 0.324$ while $\hat{T}_{\text{doub}} = 0.244$. There is a big difference in $\hat{T}$ between the single CV and double CV strategies. Notice that for split 6, single CV chooses the optimal model, m_{10}. When we check the models chosen by double CV at stage 1 for split 6, m_{10} is chosen by 6 folds, m_7 is chosen by 2 folds, and m_8 and m_{12} are chosen by 1 fold each. Since m_8 and m_{12} are sub-optimal models, they badly affect the predictive performance for the double CV strategy.

The estimated RMSE results for the three CV strategies based on NN are summarized in rows 1-3 of Table 2.4. We can see from the table that the estimated biases for the three CV strategies are all negative, i.e., they all underestimate T_{opt}. Single CV has the smallest bias while double CV has the biggest bias. In terms of the standard errors, repeated and averaged CV outperform single and double CV while the latter two CV strategies have similar performance. Overall, the repeated and averaged CV has the smallest RMSE and double CV has the biggest RMSE.

Table 2.4: Estimated RMSE for the three 10-fold CV strategies for NN and RF models and the simulated data

Method	T_{opt}	CV strategies	$\hat{T}$	$\widehat{\text{Bias}}$	SE	$\widehat{\text{RMSE}}$
NN	0.324	Single	0.317	-0.007	0.0255	0.0264
		Repeated and averaged	0.301	-0.023	0.0026	0.0231
		Double	0.281	-0.043	0.0253	0.0499
RF	0.598	Single	0.447	-0.151	0.0244	0.153
		Repeated and averaged	0.445	-0.153	0.0026	0.153
		Double	0.441	-0.157	0.0252	0.159

Next we will estimate $P_r(m_{\text{opt}}$ is chosen), for the three CV strategies.

Recall that from Table 2.2, T_{opt} is obtained at m_{10}. For the repeated

and averaged CV strategy, $\hat{m}_{\text{opt}}$ is only found once and equals m_{10}. Thus, $\hat{P}_r(m_{\text{opt}} \text{ is chosen}) = 1$. Furthermore, we can see that the best four models in terms of T are m_{10}, m_7, m_{12} and m_{11} and they are also the best four models in terms of $\hat{T}_{\text{rep}}$. Thus using repeated and averaged CV strategy would produce a similar ranking of the candidate models as those based on T.

For the single 10-fold CV strategy, $P_r(m_{\text{opt}} \text{ is chosen})$ is estimated by the proportion of times m_{10} is chosen among 100 data splits. For the double 10-fold CV strategy, CV is implemented for 10 splits of the data. At the first stage of the CV procedure, for each split, an optimal model is chosen for each fold thus $P_r(m_{\text{opt}} \text{ is chosen})$ can be estimated by the proportion of times m_{10} is chosen among $10 \times 10 = 100$ selections. Table 2.5 displays the frequencies of selection for each model.

Table 2.5: Frequencies of choosing each model for single and double 10-fold CV, for NN models and the simulated data

	Frequency of choosing model							
	m_4	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	Total
Single	4	19	7	4	46	12	8	100
Double	2	17	10	0	51	14	6	100

From the table we get

$$\hat{P}_r(m_{\text{opt}} \text{ is chosen}) = 46/100 = 0.46 \text{ for single CV,}$$

and

$$\hat{P}_r(m_{\text{opt}} \text{ is chosen}) = 51/100 = 0.51 \text{ for double CV.}$$

If we treat the best four models m_{10}, m_{11}, m_{12}, and m_7 as equivalently good

models and the remaining models as sub-optimal, we get

$$\hat{P}_r(\text{Good model is chosen}) = (19 + 46 + 12 + 8)/100 = 0.85 \text{ for single CV,}$$

and

$$\hat{P}_r(\text{Good model is chosen}) = (17 + 51 + 14 + 6)/100 = 0.88 \text{ for double CV.}$$

The above results for NN models indicate that repeated and averaged CV is the best strategy among the three CV procedures since it has the smallest mean squared error in estimating T_{opt} and the biggest probability for choosing an optimal or good model. Single and double CV both sometimes select sub-optimal models.

Now we study further the bias of $\hat{T}_{\text{sing}}$ in estimating T_{opt} for single CV. In general, note that

$$E(\hat{T}_{\text{sing}}) = \sum_i E(\hat{T}_{\text{sing}}|m_i \text{ is chosen}) P_r(m_i \text{ is chosen}),$$

whereupon the bias of $\hat{T}_{\text{sing}}$ can be written as

$$\begin{aligned} E(\hat{T}_{\text{sing}}) - T_{\text{opt}} &= \sum_i [E(\hat{T}_{\text{sing}}|m_i) - T_{\text{opt}}] P_r(m_i) \\ &= \sum_i [E(\hat{T}_{\text{sing}}|m_i) - E(\hat{T}_i) + E(\hat{T}_i) - T_i + T_i - T_{\text{opt}}] P_r(m_i) \\ &= \sum_i [E(\hat{T}_{\text{sing}}|m_i) - E(\hat{T}_i)] P_r(m_i) + \sum_i [E(\hat{T}_i) - T_i] P_r(m_i) \\ &\quad + \sum_i [T_i - T_{\text{opt}}] P_r(m_i) \\ &= \text{(I)+(II)+(III)}, \end{aligned} \tag{2.9}$$

where (I) $= \sum_i [E(\hat{T}_{\text{sing}}|m_i) - E(\hat{T}_i)] P_r(m_i)$ and is called selection bias, (II) $= \sum_i [E(\hat{T}_i) - T_i] P_r(m_i)$ and is called sample size bias and (III) $=$

$\sum_i (T_i - T_{\text{opt}}) P_r(m_i)$ is called sub-optimal model bias. Each bias is a summation of the contribution from each model. For each chosen model, the estimate of $E(\hat{T}_{\text{sing}}|m_i)$ is given in row 1 of Table 2.6 .

Table 2.6: Estimate of $E(\hat{T}_{\text{sing}}|m_i)$, i.e., conditional on choosing m_i, for single CV, for NN and RF models and the simulated data. "NA" indicates "not available" because the model was never chosen.

Chosen model	m_4	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}
NN	0.325	0.319	0.304	0.307	0.317	0.319	0.328		
RF	NA	NA	NA	NA	0.436	NA	NA	0.450	0.431

From Table 2.6, we see that $\hat{E}(\hat{T}_{\text{sing}}|m_4) = 0.325$; from Table 2.2, we see that $\hat{E}(\hat{T}_4) = 0.247$, $T_4 = 0.282$, and $T_{opt} = 0.324$; and from Table 2.5 we see $\hat{P}_r(m_4) = 0.04$. Thus the selection bias from model m_4 is: $0.325 - 0.247 = 0.078$, the sample size bias from the model is: $0.247 - 0.282 = -0.035$, and the sub-optimal model bias from the model is: $0.282 - 0.324 = -0.042$. Adding up the analogous biases of each chosen model, weighted by the probabilities of selecting the models, we get total selection bias 0.033, total sample size bias -0.026, and total sub-optimal model bias -0.014. The results are summarized in row 1 of Table 2.7. We can see that the positive selection bias partially cancels the other two negative biases and helps to reduce the total bias.

For repeated and averaged CV, the bias decomposition has the same form as (2.9), but the selection bias is 0. Hence the bias for $\hat{T}_{\text{rep}}$ is the summation of the sample size bias and the sub-optimal model bias. For NN models and the simulated data, the model chosen is the true optimal model, m_{10}, therefore, the sub-optimal model bias is also zero. Hence, the total bias is just the sample size bias which is -0.023. The bias decomposition is summarized in row 2 of Table 2.7. We can see that the sample size biases for

Table 2.7: Bias decomposition for single CV and repeated and averaged CV, for NN and RF models and the simulated data

		Bias			
Method	CV strategy	Selection	Sample size	Sub-optimal model	Total
NN	Single	0.033	-0.026	-0.014	-0.007
	Repeated	0	-0.023	0	-0.023
RF	Single	0.006	-0.131	-0.027	-0.152
	Repeated	0	-0.124	-0.029	-0.153

single CV and repeated and averaged CV are almost same. Paradoxically, single CV appears to perform well in terms of bias here, because it has *three* sources of bias and they happen to nearly cancel.

2.4.3 Results based on random forest (RF)

In this section we assess the random forest (RF) method for the three CV strategies and the simulated data. A random forest, developed by Breiman (2001), is an ensemble of many decision trees built from random sub samples of the training data, and using random subsets of the available variables during the building of each decision tree. Together the decision trees of the forest represent a model where each decision tree votes for the classification of an object, and the majority wins the vote.

A random forest is typically made up of tens or hundreds of trees. Each tree is built from a different random subset of all the available training data. Importantly though, in building each tree, a different subset of the available variables is used at each and every node to choose the variable to partition the data. With so much randomness, substantial performance gains and considerable robustness to noise, outliers, and over-fitting are achieved when compared to a single tree classifier.

There are some tuning parameters we need to choose when building a

RF model. "mtry", the number of variables which are random sampled at each node is believed to be the most important parameter. In addition, "nodesize", the minimum size of a terminal node may also influence the performance of a RF model. Setting this number larger causes smaller trees to be grown and thus take less time.

In this paper we will use different RF models with different mtry and nodesize values for the simulated data, where mtry=5, 10, 20, 50 and 100, and nodesize=1, 5 and 10. The models used are summarized in Table 2.8.

Table 2.8: RF Models for the simulated data and for AID362, Burden numbers

mtry		nodesize		
Simulated data	AID362	1	5	10
5	3	m_1	m_2	m_3
10	5	m_4	m_5	m_6
20	10	m_7	m_8	m_9
50	15	m_{10}	m_{11}	m_{12}
100	20	m_{13}	m_{14}	m_{15}

For each model, Table 2.9 displays the average value of $\hat{T}$ over 100 splits for repeated and averaged 10-fold CV and T. The standard error of the average $\hat{T}$ and the Monte Carlo error in estimating T are given in brackets. We can see from Table 2.9 that $T_{\text{opt}} = 0.598$, which is obtained at m_{14}. In terms of T, the best four models are m_{14}, m_{13}, m_{10} and m_{11} and they are also the best four models in terms of $\hat{T}_{\text{rep}}$. RMSE_{rep} is estimated using (2.7) and (2.8).

For the single 10-fold CV strategy, $\text{RMSE}_{\text{sing}}$ is estimated using (2.5) and (2.6) for 100 random splits of the data. Column 4 of Table 2.3 shows the results for 10 (of the 100) splits of the data. The model number in brackets is the model chosen for each split.

We run double 10-fold CV for 10 random splits of the data and the

Table 2.9: True performance, T, and estimated performance, $\hat{T}$, from 10-fold CV averaged over 100 splits, for RF models and the simulated data

Estimated/True	mtry	nodesize=1	nodesize=5	nodesize=10
$\hat{T}$	5	0.204 (0.0017)	0.193 (0.0017)	0.176 (0.0017)
T		0.206 (0.0007)	0.189 (0.0006)	0.156 (0.0004)
$\hat{T}$	10	0.316 (0.0022)	0.280 (0.0023)	0.241 (0.0020)
T		0.393 (0.0010)	0.347 (0.0010)	0.255 (0.0008)
$\hat{T}$	20	0.377 (0.0024)	0.340 (0.0025)	0.289 (0.0024)
T		0.489 (0.0012)	0.406 (0.0010)	0.391 (0.0010)
$\hat{T}$	50	0.422 (0.0024)	0.387 (0.0023)	0.340 (0.0024)
T		0.556 (0.0010)	0.521 (0.0011)	0.442 (0.0010)
$\hat{T}$	100	0.445 (0.0026)	0.420 (0.0024)	0.378 (0.0027)
T		0.569 (0.0010)	0.598 (0.0010)	0.509 (0.0010)

results are shown in column 5 of Table 2.3 , where the 10 splits are the same for double and single 10-fold CV.

From columns 4 and 5 of Table 2.3 , we see that for the displayed 10 splits of CV, the models chosen by single CV are m_{10}, m_{13} and m_{14} which are the best model or good models. The optimal $\hat{T}$ results for single and double CV show close agreement.

The estimated RMSE results for the three CV strategies based on RF are summarized in rows 4-6 of Table 2.4. We can see from the table that all the three CV strategies underestimate T_{opt} and their biases are much bigger than those based on NN. Single CV and repeated and averaged CV have similar biases while the bias for double CV is slightly bigger. In terms of the standard errors, repeated and averaged CV outperforms single CV and double CV, while the latter two CV strategies have pretty similar performance. Overall, repeated and averaged CV and single CV have almost the same RMSE, and double CV has a slightly bigger RMSE. Note that for the RF method, bias dominates in RMSE.

Now we estimate P_r(Good models are chosen) for the three CV strategies, treating the best four models in terms of T as equivalently good models.

For the repeated and averaged CV strategy,

$$\hat{P}_r(\text{Good model is chosen}) = \hat{P}_r(m_{10} \text{ or } m_{11} \text{ or } m_{13} \text{ or } m_{14} \text{ is chosen}) = 1.$$

Table 2.10 displays the single CV and double CV frequencies of selection for each chosen model. From the table we see

$$\hat{P}_r(\text{Good model is chosen}) = 1 \text{ for both strategies.}$$

The above results for RF models indicate that although all three CV strategies select an optimal or good model, single CV and repeated and averaged CV are slightly better than double CV in terms of mean squared error criterion.

Comparing the results for RF models with those for NN models we find that CV gives much larger bias for RF than for NN, but CV shows that RF outperforms NN because there is an even larger difference in true performance.

The simulation results for NN and RF models show that the good models selected by repeated and averaged CV agree with those chosen in terms of T. This suggests we can select good models using this CV strategy in practice, when T is unknown.

The bias decomposition for single CV and for repeated and averaged CV can be done as we did for the NN method in Section 2.4.2. The bias results are summarized in rows 3 and 4 of Table 2.7. From the table we can see that for the RF method, the selection bias is positive but smaller, the sub-optimal bias is negative and small, and the negative sample size bias dominates in the total bias.

Table 2.10: Frequencies of choosing each model for single and double 10-fold CV, for RF models and the simulated data

	Frequency			
	m_{10}	m_{13}	m_{14}	Total
Single	7	82	11	100
Double	16	70	14	100

2.5 PubChem AID362 Example

In this section we will assess NN and RF models for a real data set: the AID362 assay from PubChem (`http://pubchem.ncbi.nlm.nih.gov`). The data are also available at ChemModLab (`http://eccr.stat.ncsu.edu/ChemModLab`) and were described by Hughes-Oliver et al. (2007). The AID362 assay has 4275 compounds; for a binary response 60 are active. The ChemModLab website also gives the values of 24 descriptor variables called Burden numbers.

2.5.1 Results based on NN

For the NN method, we use the same values of size and decay as for the simulated data mentioned in Table 2.1. We cannot compute the true predictive AHR, T, or MSE for the real data. Estimated AHR, $\hat{T}$, results for repeated and averaged 10-fold CV for each model and 100 splits of the data are given in Table 2.11. The standard error of the average $\hat{T}$ for each model is given in brackets. From Table 2.11 we see

$$\hat{E}(\hat{T}_{\text{rep}}) = 0.198, \quad \text{SE}(\hat{T}_{\text{rep}}) = 0.0035.$$

We also see that m_{11} is the estimated optimal model leading to $\hat{T}_{\text{rep}}$ and the best three models are m_{11}, m_5, m_8. Although we cannot compute T for each model and hence m_{opt} here, based on the conclusions of the simulation

study in Section 3, we believe that choosing good models based on repeated and averaged CV is appropriate.

Table 2.11: NN models for the AID362 assay and Burden numbers: Average $\hat{T}$ from repeated and averaged 10-fold CV over 100 data splits. The standard error of the average is given in parentheses

Model	decay=0.1	decay=0.01	decay=0.001
size=4	0.073 (0.0007)	0.157 (0.0030)	0.140 (0.0029)
size=5	0.074 (0.0008)	0.169 (0.0031)	0.136 (0.0028)
size=7	0.074 (0.0008)	0.179 (0.0030)	0.132 (0.0025)
size=9	0.074 (0.0008)	0.198 (0.0035)	0.129 (0.0023)

Results for single 10-fold CV and double 10-fold CV for 10 splits of the data are shown in columns 2 and 3 of Table 2.12. The model number in brackets is the model chosen for each split by single CV.

Table 2.12: NN and RF models for the AID362 assay and Burden numbers: optimal $\hat{T}$ from single and double 10-fold CV for 10 data splits

	NN		RF	
Split	Single	Double	Single	Double
1	0.210 (m_{11})	0.213	0.281 (m_{13})	0.269
2	0.205 (m_5)	0.168	0.304 (m_1)	0.260
3	0.211 (m_{11})	0.193	0.314 (m_7)	0.297
4	0.256 (m_5)	0.200	0.292 (m_4)	0.295
5	0.195 (m_8)	0.154	0.226 (m_8)	0.189
6	0.235(m_3)	0.214	0.318 (m_1)	0.304
7	0.195 (m_5)	0.191	0.286 (m_1)	0.286
8	0.210 (m_{11})	0.196	0.296 (m_{13})	0.291
9	0.209 (m_8)	0.161	0.218 (m_{13})	0.204
10	0.134 (m_5)	0.170	0.244 (m_{10})	0.235

From column 2 of Table 2.12, we find that single CV would sometimes

choose sub-optimal models if it is run for just one split of the data. For example, for split 6, a sub-optimal model, m_3 is chosen. However, the $\hat{T}$ value of 0.235 is much bigger than 0.140, the average from repeated and averaged CV for m_3, i.e., m_3 was chosen because, by chance, its $\hat{T}$ value for this split far exceeded the average $\hat{T}$. Moreover, even if a good model is chosen, its estimated predictive performance could be fairly poor, e.g., for split 10, m_5 is selected, and $\hat{T}$ of 0.134 substantially underestimates the value of 0.169 in Table 2.11. A sub-optimal model is chosen for 15 out of the 100 splits for single CV.

Double CV with one split of the data can also perform poorly. In column 3 of Table 2.12, for split 5, the $\hat{T}$ value of 0.154 is small. When we check the models chosen for the 10 folds of this split, we find that m_{11} and m_5 are each selected for 4 folds respectively, m_8 is selected for 1 fold, and m_2 is selected for 1 fold. Thus choosing the very sub-optimal model m_2 is the main reason for this bad result. Over the 10 splits, double CV chooses a sub-optimal model in 12 out of 100 model selections.

For single CV, we estimate $E(\hat{T}_{\text{sing}})$ and $\text{Var}(\hat{T}_{\text{sing}})$ based on 100 random splits of the data, i.e., extending Table 2.12. For double CV, we estimate $E(\hat{T}_{\text{doub}})$ and $\text{Var}(\hat{T}_{\text{doub}})$ based on the 10 random splits shown. The results for the three CV strategies are summarized in the first three rows of Table 2.13. We can see that the estimated mean of $\hat{T}$ for single CV ($\hat{E}(\hat{T}_{\text{sing}})$) is slightly better than those for repeated and averaged CV ($\hat{E}(\hat{T}_{\text{rep}})$) and double CV ($\hat{E}(\hat{T}_{\text{doub}})$), while repeated and averaged CV has much smaller standard error than the other two CV strategies.

2.5.2 Results based on RF

For the RF method, the models used are summarized in Table 2.8. Estimated AHR, $\hat{T}$, results for repeated and averaged 10-fold CV for each model and 100 splits of the data are given in Table 2.14, with the standard error

Table 2.13: NN and RF models for the AID362 assay and Burden numbers: Average $\hat{T}$, where $\hat{T}$ is given by $\hat{T}_{\text{sing}}$, $\hat{T}_{\text{rep}}$, or $\hat{T}_{\text{doub}}$

Method	CV strategies	Average $\hat{T}$	SE
NN	Single	0.208	0.0332
	Repeated and averaged	0.198	0.0035
	Double	0.186	0.0214
RF	Single	0.269	0.0305
	Repeated and averaged	0.257	0.0031
	Double	0.263	0.0405

in brackets. From the table we see

$$\hat{E}(\hat{T}_{\text{rep}}) = 0.257, \quad \text{SE}(\hat{T}_{\text{rep}}) = 0.0031.$$

We also see that m_4 is the optimal model leading to $\hat{T}_{\text{rep}}$ and the best five models are $m_4, m_1, m_7, m_{10},$ and m_{13} which have very similar estimated preformances.

Table 2.14: RF models for the AID362 assay and Burden numbers: Average $\hat{T}$ from repeated and averaged 10-fold CV over 100 data splits. The standard error of the average is given in parentheses

Model	nodesize=1	nodesize=5	nodesize=10
mtry=3	0.255 (0.0031)	0.233 (0.0029)	0.209 (0.0027)
mtry=5	0.257 (0.0031)	0.231 (0.0029)	0.208 (0.0027)
mtry=10	0.256 (0.0030)	0.231 (0.0028)	0.206 (0.0027)
mtry=15	0.256 (0.0031)	0.230 (0.0028)	0.200 (0.0025)
mtry=20	0.255 (0.0031)	0.223 (0.0029)	0.195 (0.0025)

Results for single CV and double CV for 10 splits of the data are shown in columns 4 and 5 of Table 2.12. The model number in brackets is the

model chosen for each split by single 10-fold CV.

From column 4 of Table 2.12, we find that single CV for RF would also sometimes choose sub-optimal models if CV is run for just one split of the data. For example, for split 5, a sub-optimal model, m_8, is chosen. A sub-optimal model is chosen for 4 out of 100 splits for single CV.

Double CV with one split of the data can also perform poorly. In column 5 of Table 2.12, for split 5, the $\hat{T}$ value of 0.189 is small. When we check the models chosen for the 10 folds of this split, we find that m_1 is selected for 5 folds, m_4 is selected for 2 folds, and m_5, m_{10}, m_{14} are each selected for 1 fold respectively. Thus choosing the sub-optimal models m_5 and m_{14} is the main reason for this bad performance. Over the 10 splits, double CV chooses a sub-optimal model in 6 out of 100 model selections.

For single CV, we estimate $E(\hat{T}_{\text{sing}})$ and $\text{Var}(\hat{T}_{\text{sing}})$ based on 100 random splits of the data, i.e., extending Table 2.12. For double CV, we estimate $E(\hat{T}_{\text{doub}})$ and $\text{Var}(\hat{T}_{\text{doub}})$ based on the 10 random splits shown. The results for the three CV strategies are summarized in the last three rows of Table 2.13. We can see that the estimate of the mean $\hat{T}$ from single CV is slightly larger than those for repeated and averaged CV and double CV, while repeated and averaged CV has much smaller standard error than the other two CV strategies.

Comparing the results for RF models with those for NN models we find that all three CV strategies favor RF over NN. For both models, single CV appears to over-estimate performance despite occasional selections of sub-optimal models. Double CV also selects sub-optimal models. Its performance is quite variable, as shown by the standard errors in Table 2.13.

Although we are unable to compute the true optimal performance , T_{opt}, and hence RMSE for the real data, the single CV and double CV strategies appear to encounter the problems seen for the simulated data.

2.6 Conclusions and Discussions

This paper compares three 10-fold CV strategies in terms of their predictive performance. The results are based on finite samples from a simulation and a real data example. Two modelling methods, NN and RF, are used. Similar conclusions can be drawn for both simulation and real data, but the results for NN or RF tell a slightly different story.

For the NN method, single CV has the smallest bias among the three CV strategies in the simulation study because $\hat{T}$ usually underestimates T for any fixed model (negative sample size bias) and the positive selection bias partially cancels the bias. However, the relatively large variation between the CV estimates from one execution of CV to another would often result in the selection of sub-optimal models. For other data sets, we conjecture that the three bias terms will each have the signs seen here, but the degree of cancelation of the bias will vary.

Although double CV needs intensive computing to obtain $\hat{T}$, it fails to outperform the other strategies. It has the biggest bias and a similar variance to that for single CV. We speculate the reason is that for finite samples, especially for rare target data sets, the sample size for model selection is excessively reduced: under 10-fold CV, models fit to only 80% of the data are the basis for model selection. Thus, the chance of choosing sub-optimal models increases, and predictive performance is affected.

For the RF method, the differences among the three CV strategies are not very substantial for the simulated example in terms of MSE of estimating predictive performance and the probability of choosing a good model. The selection bias is much smaller than that for NN method. We have similar findings for the simulated and real data. In general, the RF method is more robust in these senses than the NN method and has better predictive performance. The RF algorithm averages over many trees which is analogous to the repeated and averaged CV strategy. Note that the RF method can use the out-of-bag sample to estimate prediction error. Empirical results

not reported here show that the out-of-bag error estimates are quite similar to what we have obtained in Section 2.4.3.

Repeated and averaged CV is the best strategy since it produces estimate of predictive performance with moderate bias and smallest variances. When enough data splits are made, the probability of choosing an optimal model is high. It is a very computing intensive strategy, however. If the number of the candidate models is huge or fitting some candidate model takes a long time, then implementing this CV strategy for 100 splits of the data (as we did in this paper) is not computationally feasible. In this situation we could use, say, 10 splits of the data and the standard error of the predictive performance (e.g., AHR) could be used to guide whether further splits are needed. The number of fits per model can also be reduced by a data-adaptive repeated and averaged CV algorithm, developed by Shen, Welch and Hughes-Oliver (2008), where optimal models are found through a sequential approach.

In this chapter, MSE was used as one criterion to assess the predictive performance of the three CV strategies. Although MSE would be sensitive to outliers, it has the advantage of a straightforward bias-variance decomposition.

This chapter presents results only for 10-fold CV, but we have computational results for 5-fold and 20-fold CV. We find that $\hat{T}$ for any of the three CV strategies increases with the number of folds. Increasing v means that a larger proportion of the data are used for model training and hence model selection, but the computational burden increases.

Bibliography

[1] Breiman, L. (2001). Random Forest. *Machine Learing* 45(1), 5–32.

[2] Burman, P. (1989). A Comparative Study of Ordinary Cross-Validation, v-Fold Cross-Validation and the Repeated Learning-Testing Methods. *Biometrika* 76, 503–514.

[3] Dietterich, T.G. (1998). Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Computation* 10, 1895–1923.

[4] Dudoit, S. and van der Laan, M.J. (2003). Asymptotics of Cross-Validated Risk Estimation in Estimator Selection and Performance Assessment. Technical Report 126, Division of Biostatististics, University of California, Berkeley. URL `www.bepress.com/ucbbiostat/paper126`.

[5] Efron, B. (2004). The Estimation of Prediction Error: Covariance Penalties and Cross-Validation. *Journal of the American Statistical Association* Vol. 99, No. 467, Theory and Methods, 619–642.

[6] Hawkins, D.M., Basak, S.C., Mills, D. (2003). Assessing Model Fit by Cross-Validation. *J. Chem. Inf. Comput. Sci.* 43, 579–586.

[7] Hughes-Oliver, J. M., Brooks, A. D., Welch, W. J., Khaledi, M. G.,Hawkins, D., Young, S. S., Patil, K., Howell, G. W., Ng, R. T., and Chu, M. T. (2007). ChemModLab: A Web-Based Cheminformatics Modeling Laboratory. In review.

[8] Li, K.-C. (1987). Asymtotic optimality for c_p, c_l, cross-validation and generalized cross-validation: Discrete index set. *Annals of Statistics* 15:958–975.

[9] Mosteller, F. and Tukey, J. W. (1977). *Data Analysis and Regression.* Addison-Wesley Publishing Company.

[10] Ripley, B. D. (1996). *Pattern Recognition and Neural Networks.* Cambridge: Cambridge University Press.

[11] Shao, J. (1993). Linear Model Selection by Cross-Validation. *Journal of the American Statistical Association* Vol. 88, No. 422, Theory and Methods, 486–494.

[12] Shen, H., Welch W.J., Hughes-Oliver, J. M. (2008). Efficient, Adaptive Cross Validation for Tuning and Comparing Models. Manuscript.

[13] Sinisi, S.E. and van der Laan, M.J. (2004). Loss-Baseed Cross-Validated Deletion/Substitution/Addition Algorithms in Estimation. Technical Report 143. Division of Biostatististics, University of California, Berkeley. URL `www.bepress.com/ucbbiostat/paper143`.

[14] Stone, M. (1974). Cross-Validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society* Series B, 36, 111–147.

[15] Stone, M. (1977). Asymptotics for and against cross-validation. *Biometrika* 64(1): 29–35.

[16] Wang, Y. (2005). Statistical Methods for High Throughput Screening Drug Discovery Data. PhD thesis, Department of Statistics and Actuarial Science, University of Waterloo, Waterloo, Ontario, Canada.

[17] Zhang, P. (1993). Model Selection Via Multifold Validation. *The Annals of Statistics* Vol. 21, No. 1, 299–313.

Chapter 3

Efficient, Adaptive Cross Validation for Tuning and Comparing Models

3.1 Introduction

The application area that motivated this research illustrates the enormous computational burden that can occur when cross validation (CV) is used to tune and select statistical models. Our Exploratory Center for Cheminformatics Research (`http://eccr.stat.ncsu.edu`), funded by the National Institutes of Health Roadmap for Medical Research, is comparing statistical modeling methods on assay data from PubChem (`http://pubchem.ncbi.nlm.nih.gov`). For a given assay, activity (the response variable) against a particular biological target is measured for thousands or tens of thousands of drug-like molecules. Several high-dimensional sets of chemical descriptors (explanatory variables) are available to characterize the chemical properties of the molecules. A statistical model attempts to relate biological activity to the chemical descriptors as part of drug discovery. Currently, for each assay, the web-based Cheminformatics Modeling Laboratory or ChemModLab (`http://eccr.stat.ncsu.edu/ChemModLab`) compares, via CV, 16 statisti-

A version of this chapter will be submitted for publication. Authors: Shen H., Welch W. J., Hughes-Oliver J. M.

cal methods, many of which are computationally demanding, and five candidate sets of descriptor variables. Thus, $16 \times 5 = 80$ modeling strategies are assessed and compared on data sets with thousands of observations and high-dimensional explanatory variables.

Moreover, ideally each of these 80 strategies should be tuned with respect to one or more user-selected parameters, greatly increasing the number of candidate models to be compared. For example, a neural network has several user-defined tuning parameters controlling the network complexity, such as the number of hidden units and a decay parameter. If many sets of values for the tuning parameters are tried, potentially hundreds or thousands of computationally demanding models need to be compared for the large PubChem data sets.

CV (Stone 1974) is widely used for this type of study, albeit usually on a much smaller scale. In a 10-fold cross-validation, for example, the observations are split into 10 groups or folds, one group is considered as test data for assessing prediction accuracy, and the other nine groups are used for model fitting. This process is repeated with each of the groups in turn as test data. Thus, further increasing the computational burden already described, a fixed model (a statistical model with given values of all tuning parameters and a descriptor set) has to be fitted 10 times.

There is yet another addition to the computational challenge. CV is based on a random split of the data, and, as we illustrate in Section 3.2, there can be considerable variation from one split to another. Thus, numerous data splits may be necessary to compare models reliably.

Thus, the overall computational effort appears to be simply infeasible for the comprehensive comparisons we have outlined for large PubChem assay data sets. To our knowledge, currently all comparisons of this type hence have some degree of unreliability and/or sub-optimality, due to randomness in CV and lack of effective tuning, respectively.

Much theoretical work has been done on CV. Stone (1974, 1977) focused

mainly on properties for leave-one-out (or n-fold) CV. Li (1987), Shao (1993) and Zhang (1993) investigated v-fold CV procedures for linear models and general v. Burman (1989) established theoretical results for v-fold CV for a wider class of models. More recently, Dudoit and van der Laan (2003) derived asymptotic properties for a broad definition of CV (e.g., leave-one-out, v-fold, Monte Carlo, etc.) for model selection and performance assessment, and Yang (2006) established the consistency of CV for classification. The theoretical developments parallel the extremely wide use of CV by researchers for assessing and selecting models, for example, Dietterich (1998), Hawkins et al. (2003), Sinisi and van der Laan (2004), and Hughes-Oliver et al. (2007).

In this article, we will focus on 10-fold CV, though the methodology applies to v-fold CV for any feasible v. We propose a data-adaptive approach involving multiple repeats of CV for the candidate models. At any stage, the CV analyses available from repeated data splits are used to perform a multiplicity-adjusted statistical test to eliminate all candidate models that are inferior to at least one other. Only those models that survive move on to the next stage and have a further CV performed to increase the test power based on a new, common data split. In this way, during model tuning, very poor settings of the tuning parameters are quickly dismissed and computational effort is concentrated on the best settings. The search terminates when one setting emerges as the winner, or when the differences in performance between the surviving settings are practically unimportant with some statistical confidence. A similar approach is used to compare optimized models. In the PubChem application there will be one optimized model for each statistical modeling strategy, i.e., a class of models such as k-nearest neighbors with one of the available descriptor sets. It is also possible to combine tuning with comparison across optimized models in one dynamic search.

Secondly, we develop more efficient tests for comparing models. This ex-

tends the idea of matching by using the same data splits across CV analyses (e.g., Dietterich, 1998). By matching at the level of individual observations rather than data split, moderate differences in performance between models can sometimes be detected with just one set of CV analyses from one data split. Thus, poor performers are potentially eliminated with a minimum of computing.

Overall, the aim of this article is to develop a sequential approach for comprehensive and reliable model tuning and selection via CV. In particular, for the PubChem applications, users of ChemModLab will have automatic comparison of a vast number of tuned modeling strategies, with a reasonable turn-around time.

Related to our sequential tests via CV, Maron and Moore (1997) developed a "racing" algorithm to test a set of models in parallel. The algorithm sequentially increases data points to build and test candidate models before using all of the data. In their paper, leave-one-out CV was used to compute the prediction error. In contrast, our algorithms use all the data points at all stages, 10-fold CV is implemented to estimate the prediction error, and computational speed-up is achieved by reducing the number of models.

The paper is organized as follows. In Section 3.2 we describe a typical PubChem data set and the performance assessment measures relevant to the application. In Section 3.3, we illustrate that there may be substantial variation in CV performance estimates from one random data split to another, requiring multiple data splits for reliable comparison. Section 3.4 describes three data-adaptive algorithms for sequentially comparing models. Whereas Section 3.4 is focused on tuning a given modeling strategy, i.e., a given statistical method and set of data, Section 3.5 considers tuning *and* comparisons across qualitatively different modeling strategies, i.e., different types of statistical models and/or different explanatory variable sets. The PubChem data set is used throughout for illustration. Finally, some conclusions are presented in Section 3.6.

3.2 PubChem AID362 Data and Assessment Measures

ChemModLab (Hughes-Oliver et al., 2007) catalogs the data for five assays: AID348, AID362, AID364, AID371, and AID377. ("AID" stands for "Assay ID".) In this paper, we will focus on AID362, a formylpeptide receptor ligand binding assay that was conducted by the New Mexico Molecular Libraries Screening Center; the same CV comparison methodologies would be applied independently to other assays in PubChem.

AID362 has assay data for 4,275 molecules. Various responses are available, but here we work with a binary inactive/active (0/1) measure. Of the 4,275 molecules, only 60 were assayed to be active. Via computational chemistry, ChemModLab generates five sets of descriptor (explanatory) variables: Burden numbers, pharmacophore fingerprints, atom pairs, fragment fingerprints, and Carhart atom pairs, with 24, 121, 395, 597, and 1,578 variables, respectively.

The purpose of building a statistical model here is to predict the AID362 inactive/active assay response from the descriptor variables. Note that the descriptor variables are produced by *computational* chemistry. Thus, it is feasible to compute them cheaply for vast numbers of compounds in a chemical library or even in a virtual library of chemical formulas for molecules that have not yet been synthesized. The aim of the predictive model, built from assay data for relatively few molecules, is to choose the molecules in the bigger library that are most likely to be active when assayed. Such a focused search generates "hits" for drug development more efficiently than assaying all the compounds available, even if this is feasible.

The typical rarity of active compounds and the aim of identifying a small number of promising compounds in a large library means that special predictive performance measures have been developed for modeling in drug discovery. Misclassification rate, often used for a binary response, is not appropriate, as even the useless, null model that always classifies as "inactive"

will have a high accuracy rate when active molecules are so rare. The objective is more to *rank* compounds in terms of their probability of activity, so that a sample of the desired size of the most promising compounds can be chosen from a library.

A widely used criterion is a simple function of the number of hits found, h_{300}, among 300 compounds selected using a predictive model. Specifically, suppose a predictive model generates $\hat{p}_i$, the probability that compound i among N unassayed compounds is active ($i = 1, \ldots, N$). We then order the compound indices via the permutation π such that $\hat{p}_{\pi(1)} \geq \cdots \geq \hat{p}_{\pi(N)}$. Suppose first there are no ties. The 300 compounds indexed by $\pi_{(1)}, \ldots,$ $\pi_{(300)}$ are selected for assay, and h_{300} is simply the number of actives (hits) found among them. In general, if $\hat{p}_{\pi(300)}$ ties with the $a + b$ estimated probabilities $\hat{p}_{\pi(300-a+1)}, \ldots, \hat{p}_{\pi(300+b)}$ for $a \geq 1$ and $b \geq 0$, then h_{300} is defined as

$$h_{300} = h_{300-a} + \frac{a}{a+b} h_{\text{tie}}, \tag{3.1}$$

where h_{300-a} and h_{tie} are the number of hits found among the compounds with indices $\pi(1), \ldots, \pi(300-a)$ and $\pi(300-a+1), \ldots, \pi(300+b)$, respectively. This is the expected number of hits if a compounds are randomly selected from the $a+b$ with tied probabilities to make a total of 300 selected. No ties for $\hat{p}_{\pi(300)}$ is just a special case of (3.1) with $a = 1$ and $b = 0$.

Initial enhancement (IE), used for example by Hughes-Oliver et al. (2007), is just $(h_{300}/300)/r$, where r is the activity rate in the entire collection of N compounds. Thus, it measures the rate of finding actives among the 300 chosen compounds relative to the expected rate under random selection. A good model should have IE values much larger than 1. As IE is just a linearly increasing function of h_{300}, the two criteria are equivalent, and we use the simpler h_{300} in this article. Users concerned about the arbitrariness of selecting 300 compounds may prefer the average hit rate (AHR) proposed by Wang (2005), which averages performance over all selection sizes but favors models which rank active compounds ahead of inactive compounds in terms

of $\hat{p}_i$. Algorithms 1 and 3 in Section 3.4 could be applied directly to AHR without modification.

In defining the assessment measure h_{300}, we have assumed there is a training data set to build a model and a further independent test set of N compounds available to assess it. This article is concerned with CV, however, where the same n observations are used for training and for testing. Under 10-fold CV, for instance, when a particular data fold is removed to serve as test data, the model fitted to the remaining data generates the $\hat{p}_i$ values for the compounds in that fold. After cycling through all 10 folds, the $\hat{p}_i$ values are put together so that there is a $\hat{p}_i$ for all n compounds. We then define h_{300} (or an alternative criterion) exactly as above except that we choose 300 compounds from the $n \geq 300$ instead of from an independent set of size N.

3.3 Variation in Cross Validation

We now demonstrate that there can be substantial variation in the performance estimates from 10-fold CV from one random split of the data to another, potentially requiring multiple splits for reliable model tuning or selection. For illustration, the PubChem AID362 assay data will be modeled using a neural network (NN) (see, e.g., Ripley, 1996) with one hidden layer and a variation of Burden numbers (Burden, 1989) as the descriptor set. For the AID362 assay, there are 60 active compounds among 4,275 molecules, and the Burden number descriptor set has 24 variables.

We will tune two important parameters of the NN: the number of units in the hidden layer, which controls the size or complexity of the network, and a decay parameter, where smaller values shrink the network weights less and lead again to a more complex network. In this tuning study, size takes the values 5, 7, and 9, and decay takes the values 0.1, 0.01, and 0.001. Thus, tuning will select among $3 \times 3 = 9$ models generated by all combinations of the two tuning parameters.

For each model, 10-fold CV is run for 100 random splits of the data, and

Table 3.1: Sample means of h_{300} for 10-fold CV across 100 data splits for neural networks with different values of size and decay, and Burden numbers as the descriptor set, applied to the PubChem AID362 assay data.

Model	1	2	3	4	5	6	7	8	9
size	5	5	5	7	7	7	9	9	9
decay	0.1	0.01	0.001	0.1	0.01	0.001	0.1	0.01	0.001
# of hits	18.7	31.2	28.7	18.6	30.2	30.7	18.5	30.7	31.4
S.E.	0.18	0.24	0.27	0.19	0.24	0.27	0.18	0.22	0.26
Rank	7	2	6	8	5	3	9	4	1

the histograms in Figure 3.1 show the estimated distributions of the h_{300} assessment measure defined in (3.1). We can see that there are considerable differences between the h_{300} distributions across the tuning parameters values considered, i.e., tuning is important. There is also considerable variation within a fixed set of tuning parameter values. For example, for size=5 and decay=0.01, which is one of the better performing models, h_{300} ranges from 26 to 37. We will take the population mean performance over a large number of repeated cross validations as a reliable measure of performance; reliable in the sense that random cross validation variation is eliminated. Table 3.1 displays the observed sample means of h_{300} with their standard errors. Models 2, 5, 6, 8, and 9 have better sample means than models 1, 3, 4, and 7. Moreover, the standard errors are fairly small relative to the differences between the sample means across these two groups, suggesting that the weaker performers could be dismissed with fewer than 100 random data splits, whereas finding the best parameter values among the better models will take considerable work (though perhaps not requiring 100 random data splits). This is the basic idea underlying the adaptive algorithms of Section 3.4.

Such a comparison should take into account that data split would naturally be a blocking factor. Every time a random data split is generated, all models under consideration are assessed via CV using this same split.

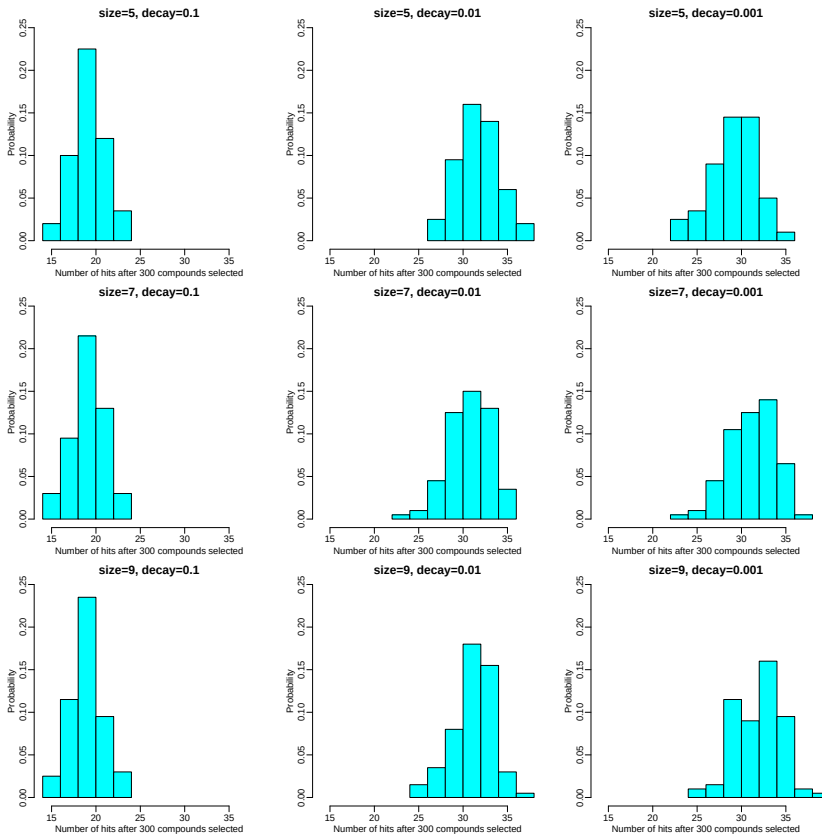

Figure 3.1: Histograms showing the distribution of h_{300} values for 10-fold CV across 100 data splits for neural networks with different values of size and decay, and Burden numbers as the descriptor set.

Thus, the 100 data splits leading to the data in Figure 3.1 are 100 blocks. Figure 3.2 shows the results for five blocks, with each line representing one split. The approximate parallelism of the curves indicates that including split as a blocking factor will lead to more powerful comparisons. Figure 3.2 also suggests that comparing models based on just one split may lead to a biased estimator of performance. For each curve, suppose we select the model (set of tuning parameter values) with the largest observed value of h_{300}. We note first that, probably due to selection bias, the h_{300} value of the winning model tends to be in the upper tail of its distribution in Fig-

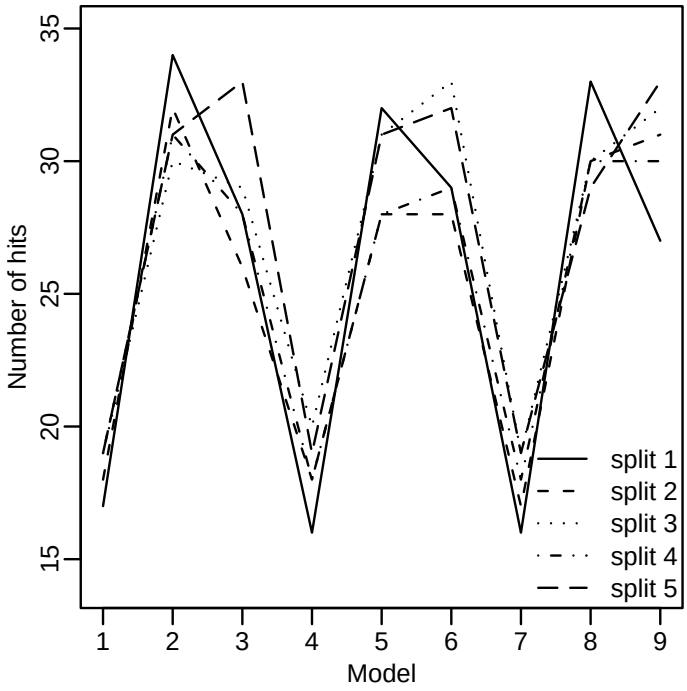

Figure 3.2: h_{300} values for 10-fold CV and five data splits (five lines) for neural networks with different values of size and decay, and Burden numbers as the descriptor set.

ure 3.1. Secondly, for the fifth split, sub-optimal model 3 has the best value of h_{300}. Thus, there is a need for multiple splits for reliable assessment and comparison.

3.4 Algorithms for Adaptive Model Search Via Sequential CV

3.4.1 Algorithm 1 (data splits as blocks)

Suppose there are m models to be compared. For much of this article, we will be comparing m sets of values for the tuning parameters of a given type of

statistical modeling method, in the context of a fixed descriptor (explanatory variable) set. Comparisons across qualitatively different statistical models and/or different sets of explanatory variables are also possible, however (Section 3.5). The algorithm will attempt to remove models sequentially until m is reduced to 1.

At each iteration, a new random data split is created for CV, and a CV estimate of performance is computed for the surviving models. Suppose there are m surviving models, and results from running 10-fold CV are available for $s \geq 2$ random splits. The assessment measure is computed for every model and split. We will use h_{300} in (3.1), but for this first version of the algorithm any user-defined measure could be employed, e.g., the average hit rate in Section 3.2 or, for a continuous response, empirical predictive mean squared error. In general, y_{ij} will denote the assessment measure for model i and data split j.

If a randomly chosen split is applied across all models, split is a a blocking factor, and we can model y_{ij} as generated by

$$Y_{ij} = \mu + \tau_i + \beta_j + \varepsilon_{ij} \qquad (i = 1, \ldots, m; j = 1, \ldots, s),$$

where μ is an overall effect, τ_i is the effect of model i, β_j is the effect of split j, and ε_{ij} for $i = 1, \ldots, m$ and $j = 1, \ldots, s$ are random errors, assumed to have independent normal distributions with mean 0 and variance σ^2. This is the model for a randomized block design, though we point out that randomization within a block, for example executing the analyses for the m models in a random order, has no relevance for such a "computer experiment".

We want to test the hypotheses

$$H_0 : \tau_i = \tau_{i'} \text{ versus } H_1 : \tau_i \neq \tau_{i'}$$

for all $i \neq i'$. For a particular pair of models indexed by i and i', rejecting

H_0 in favor of H_1 at some significance level implies that one of the models may be eliminated as inferior to the other. After removing all such dominated models, at the next iteration, further CV computational effort will be concentrated on the surviving models.

At least initially, m may be large, and a multiplicity-adjusted test is desirable. Tukey's test (Dean and Voss, 1999, Chapter 4; Montgomery, 1997, Chapter 3) is a common choice for such multiple comparisons. Let

$$\bar{y}_{i\cdot} = \frac{1}{s}\sum_{j=1}^{s} y_{ij}, \tag{3.2}$$

denote the sample mean performance over the s splits for model i ($i = 1, \ldots, m$). For any $i \neq i'$, the null hypothesis H_0 is rejected in favor of H_1 at level α if

$$|\bar{y}_{i\cdot} - \bar{y}_{i'\cdot}| > T_\alpha(m, s) = q_\alpha(m, (m-1)(s-1))\sqrt{\frac{\mathrm{MSE}(m, s)}{s}},$$

where $q_\alpha(m, (m-1)(s-1))$ is the studentized range statistic with m and $(m-1)(s-1)$ degrees of freedom, and $\mathrm{MSE}(m, s)$ is the mean square for error under a randomized-block analysis of variance with m models (treatments) and s splits (blocks). A set of simultaneous $100(1-\alpha)$ percent confidence intervals for all pairwise differences $\tau_i - \tau_{i'}$ for $i \neq i'$ is given by

$$\tau_i - \tau_{i'} \in (\bar{y}_{i\cdot} - \bar{y}_{i'\cdot} \pm T_\alpha(m, s)), \tag{3.3}$$

where

$$T_\alpha(m, s) = q_\alpha(m, (m-1)(s-1))\sqrt{\frac{\mathrm{MSE}(m, s)}{s}} \tag{3.4}$$

is the Tukey value. Other tests for multiple comparisons could be applied, such as Fisher's least significant difference test or Duncan's multiple range test, etc. (Montgomery, 1997). We use Tukey's test since it controls the overall error rate well.

Figure 3.3: Adaptive model search via sequential CV (Algorithm 1: Iterate until one model is left or a maximum of S data splits has been performed.)

1. Let m be the number of candidate models. Set $s = 1$ and make a random split of the data. Compute the CV performance measures y_{i1} for $i = 1, \ldots, m$.
2. Do the following steps while $m > 1$ and $s < S$.
   (a) Replace s by $s+1$ and compute the CV performance measure y_{is} for $i = 1, \ldots, m$.
   (b) Compute $\bar{y}_{i\cdot}$ from equation (3.2) for $i = 1, \ldots, m$.
   (c) Rank the $\bar{y}_{i\cdot}$ such that $\bar{y}_{(1)\cdot} \geq \bar{y}_{(2)\cdot} \geq \ldots \geq \bar{y}_{(m)\cdot}$.
   (d) Compute the Tukey value $T_\alpha(m, s)$ in equation (3.4).
   (e) For $i = 2, \ldots, m$, if $\bar{y}_{(1)\cdot} - \bar{y}_{(i)\cdot} > T_\alpha(m, s)$ eliminate model i.
   (f) Let m be the number of surviving models (with new indices according to Step 2c).

Figure 3.3 gives pseudo code for the above sequential algorithm. It iterates until only one model is left, subject to a maximum of S random data splits and hence S CV analyses for any model. We use $S = 100$ hereafter. Note that this algorithm needs at least two executions of CV for each initial model from two random data splits.

For illustration, we revisit the PubChem AID362 example in Section 3.3, where the descriptor set is formed from Burden numbers, and the problem is to tune the parameters decay and size for a neural network model. The nine candidate models, i.e., the nine combinations of decay and size values, were given in Table 3.1.

Table 3.2 shows the results of applying Algorithm 1 to this example. After $s = 2$ splits, the average h_{300} values for the nine models, $\bar{y}_1, \ldots, \bar{y}_9$, are

$$17.5, 33.0, 27.0, 17.0, 30.0, 28.5, 16.5, 31.5, 29.0,$$

Number	Neural network model								
of splits	1	2	3	4	5	6	7	8	9
0	✓	✓	✓	✓	✓	✓	✓	✓	✓
2		✓	✓		✓	✓		✓	✓
3		✓	✓		✓	✓		✓	✓
4		✓	✓		✓	✓		✓	✓
5		✓			✓	✓		✓	✓
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
57		✓			✓	✓		✓	✓
58		✓				✓		✓	✓
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
68		✓				✓		✓	✓
69		✓						✓	✓
70		✓							✓
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
100		✓							✓

Table 3.2: Algorithm 1 applied to tuning the values of size and decay for a neural network for the PubChem AID362 assay data with Burden numbers as the descriptor set. The models surviving after each split are denoted by a check mark.

MSE$(9, 2) = 3.39$, and the Tukey value is 7.51 for significance level $\alpha = 0.05$. Since $\bar{y}_{2\cdot} - \bar{y}_{i\cdot} > 7.51$ for $i = 1$, 4, and 7, these three models are dismissed. Recall from Table 3.1 that they are indeed the worst when averaged over 100 splits, but the sequential algorithm eliminates them after just two CV splits. Hence in Table 3.2, only models 2, 3, 5, 6, 8, and 9 survive the second split and are included for a third round of CV based on another split. After five splits, model 3 is removed. Models 5, 6, and 8 are removed after 58, 69, and 70 splits, respectively. The two remaining models, 2 and 9, are still in contention when the algorithm stops due to restricting the computational effort to 100 splits. From the average h_{300} values given in Table 3.1, we know that these two models are very similar in performance, and are hard to distinguish. This motivates Algorithm 3 in Section 3.4.3, but next we improve Algorithm 1.

3.4.2 Algorithm 2 (observations as blocks)

Algorithm 1 in Section 3.4.1 needs at least two CV data splits for every one of the m models, which may be computationally expensive if m is large. We now describe another multiplicity-adjusted test, aimed at eliminating bad models after only one CV data split.

Unlike Algorithm 1, the revised algorithm is applicable only to an assessment measure that is a sum or average of contributions from individual observations. The criterion h_{300} in (3.1) is of this form, and we continue to use it, but we note that only the active compounds in the data set can make a non-zero contribution to h_{300}, and it is sufficient to consider them only. Specifically, suppose there are $A \geq 2$ active compounds in the data set ($A = 60$ for the AID362 assay). For any given model, its CV analysis leads to estimated probabilities of activity $\hat{p}_{\pi(1)} \geq \cdots \geq \hat{p}_{\pi(n)}$ for the n compounds in the data set. We can write

$$h_{300} = \sum_{j=1}^{A} y_j^*,$$

where y_j^* is the contribution from active compound j. From the definition of h_{300} in (3.1),

$$y_j^* = \begin{cases} 1 & \text{if active compound } j \text{ is one of the first } 300-a \text{ compounds selected} \\ \dfrac{a}{a+b} & \text{if active compound } j \text{ appears among the } a+b \text{ compounds with } \hat{p} \\ & \text{tying with } \hat{p}_{\pi(300)} \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

(Recall that $\hat{p}_{\pi(300)}$ ties with the $a+b$ estimated probabilities $\hat{p}_{\pi(300-a+1)}, \ldots, \hat{p}_{\pi(300+b)}$ with $a \geq 1$ and $b \geq 0$, which includes no ties if $a = 1$ and $b = 0$.)

For example, suppose a CV analysis leads to estimated probabilities of activity $\hat{p}_{\pi(1)} \geq \cdots \geq \hat{p}_{\pi(n)}$ such that $\hat{p}_{\pi(300)}$ has the eight ties $\hat{p}_{\pi(298)}, \ldots, \hat{p}_{\pi(305)}$. Of the, say, $A = 60$ active compounds, 25 have estimated probabilities among $\hat{p}_{\pi(1)}, \ldots, \hat{p}_{\pi(297)}$; they each have $y_j^* = 1$ in (3.5) because they must each contribute one hit to h_{300}. Another two active compounds have estimated probabilities among $\hat{p}_{\pi(298)}, \ldots, \hat{p}_{\pi(305)}$; they each have $y_j^* = 3/8$, the probability of being selected 298th, 299th, or 300th when the eight selections $298, \ldots, 305$ are made in random order.

We now consider CV analyses to compare m models based on one common random data split. Let y_{ij}^* be the contribution of active compound j to h_{300} for model i and active compound j, for $i = 1, \ldots, m$ and $j = 1, \ldots, A$. A multiplicity-adjusted test parallels that in Section 3.4.1. In the randomized-block analysis, the blocks are now the A active compounds rather than data splits (there is only one). If a randomly chosen split is applied across all models, we can model y_{ij}^* as generated by

$$Y_{ij}^* = \mu^* + \tau_i^* + \beta_j^* + \varepsilon_{ij}^* \qquad (i = 1, \ldots, m; j = 1, \ldots, A),$$

where μ^* is an overall effect, τ_i^* is the effect of model i, β_j^* is the effect of active compound j, for $i = 1, \ldots, m$ and $j = 1, \ldots, A$, and ε_{ij}^* are random errors, assumed to have independent normal distributions with mean 0 and

variance σ^{*^2}. Similarly, the Tukey value in (3.4) is replaced by

$$T_\alpha(m, A) = q_\alpha(m, (m-1)(A-1))\sqrt{\frac{\text{MSE}(m, A)}{A}},$$

where the studentized range statistic q has degree of freedom m and $(m-1)(A-1)$. Analogous hypothesis tests eliminate all models significantly different from the one with the best observed performance.

If $s \geq 2$ data splits have been made, we could define blocks in terms of active compounds *and* data splits, i.e., the number of blocks would be sA. Some experimentation indicates that Algorithm 1 in Section 3.4.1 eliminates inferior models faster, however, for $s \geq 2$. Thus, for Algorithm 2 we use the Tukey test based on active compounds as blocks only for the first data split. After very poor models are eliminated, a second split is made and the data-adaptive search proceeds for the surviving models as in Section 3.4.1 with $s \geq 2$ splits as blocks.

We now revisit the example of tuning a neural network in Section 3.4.1 (the results for Algorithm 1 were presented in Table 3.2). Figure 3.4 depicts Algorithm 2's progress, plotting h_{300} versus split. After running one split of 10-fold CV, the model with size = 5 and decay = 0.01 has the largest h_{300} value. The vertical line drawn down from this value has length $T_\alpha(m, A)$, where $m = 9$ and $A = 60$. It is based on Tukey's test with the 60 active compounds as blocks. The h_{300} values for models 1, 4 and 7 fall below this line and they are eliminated with one CV split. For $2, 3, \ldots$ splits, $\bar{y}_{i\cdot}$, the average of h_{300} over the splits, is plotted for the surviving models, and the vertical lines have length $T_\alpha(m, s)$, where m is the number of models surviving to s splits. It is seen that after 2, 3, or 4 splits of CV, no further models are eliminated. After 5 splits, model 3 is dismissed, and after 9 splits models 2, 5, 6, 8 and 9 still survive. The two models with the largest h_{300} averages, models 2 and 9, are connected with a dashed line in Figure 3.4. These models are still competitors after 100 splits of CV. The vertical line drawn at 100 splits is very short; nonetheless, these models are so close

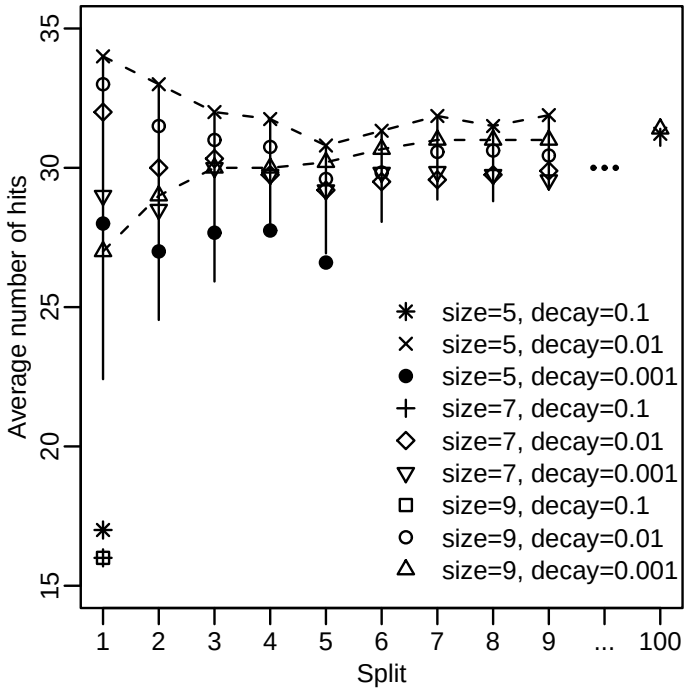

Figure 3.4: Algorithm 2 applied to tuning the values of size and decay for a neural network for the PubChem AID362 assay data with Burden numbers as the descriptor set. The two models with the largest h_{300} averages, models 2 and 9, are connected with a dashed line.

in performance that they cannot be distinguished. Again, this motivates Algorithm 3.

3.4.3 Algorithm 3 (modified stopping criterion)

As has already been illustrated, if the predictive performances for several candidate models are very similar, it can take many data splits and CV analyses to distinguish them. Particularly for model tuning, it would be more efficient to modify the stopping criterion so that the algorithm stops once it is clear that the current leading performer cannot be beaten by a practically important amount.

We implement such a stopping criterion via the confidence intervals in (3.3). Again, rank the m models surviving at any iteration in terms of their average predictive performances, i.e., $\bar{y}_{(1)\cdot} \geq \bar{y}_{(2)\cdot} \geq \ldots \geq \bar{y}_{(m)\cdot}$. Notationally, we will use s (number of splits) for the number of blocks in these averages, but the same method can be applied with observations as blocks as in Section 3.4.2. At some confidence level, we want to be sure that $\tau_{(i)} - \tau_{(1)} < p_0$ for all $i = 2, \ldots, m$, where p_0 is a given practically insignificant performance difference. From (3.3),

$$\tau_{(i)} - \tau_{(1)} \in (\bar{y}_{(i)\cdot} - \bar{y}_{(1)\cdot} \pm T_\alpha(m, s)).$$

Thus, to stop with the model giving $\bar{y}_{(1)\cdot}$ declared as the winner,

$$\bar{y}_{(i)\cdot} - \bar{y}_{(1)\cdot} + T_\alpha(m, s) < p_0 \quad (\text{for all } i = 2, \ldots, m).$$

Clearly, it is sufficient to test this condition for $i = 2$; hence the revised stopping criterion is simply

$$T_\alpha(m, s) - (\bar{y}_{(1)\cdot} - \bar{y}_{(2)\cdot}) < p_0. \tag{3.6}$$

For the example of tuning a neural network for the AID362 assay data and Burden number descriptors, the values of $T_\alpha(m, s) - (\bar{y}_{(1)\cdot} - \bar{y}_{(2)\cdot})$ in (3.6) for data splits 1–38 are:

$$10.59, 6.96, 5.08, 3.16, 3.27, 2.60, 2.14, 2.20, 1.67, \ldots, 1.05, 0.91.$$

(The hybrid observations/splits as blocks algorithm of Section 3.4.2 is being used here.) If we take $p_0 = 1$ as the practically insignificant performance difference, the algorithm stops after 38 splits of 10-fold CV, with surviving models 2, 5, 6, 8 and 9. Model 2 with size = 5 and decay = 0.01 would be declared the "tuned" model for practical purposes. If we set $p_0 = 2$, the algorithm stops after just nine splits. Models 2, 5, 6, 8 and 9 are again the

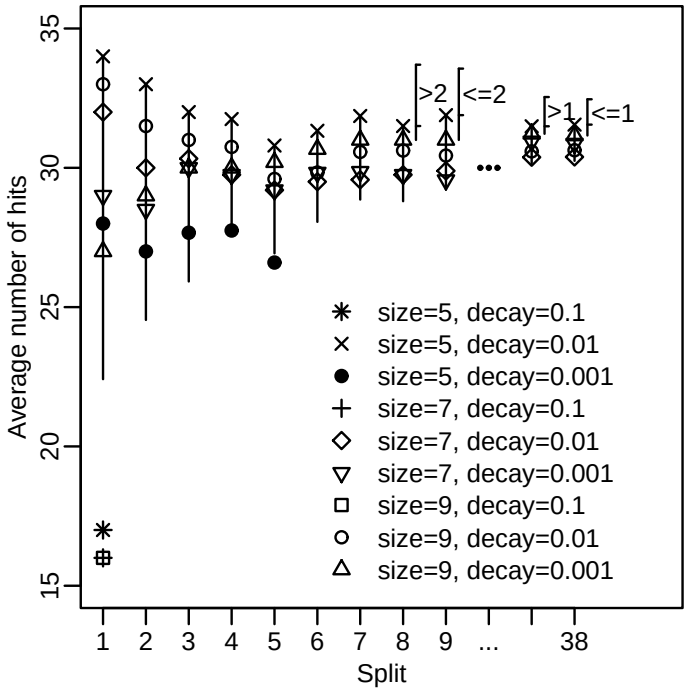

Figure 3.5: Algorithm 3 applied to tuning the values of size and decay for a neural network for the PubChem AID362 assay data with Burden numbers as the descriptor set.

survivors, and again Model 2 is declared the winner for the neural networks / Burden numbers modeling strategy. Figure 3.5 illustrates the iterations of the algorithm. In particular, the vertical lines shown to the right of the performance averages for 8, 9, 37, and 38 splits start at $\bar{y}_{(2)\cdot}$ and have length $T_\alpha(m, s)$. If they extend less than p_0 past $\bar{y}_{(1)\cdot}$ then the revised stopping criterion (3.6) is satisfied.

Recall that when we try to establish the one winning model via Algorithms 1 or 2, 100 data splits and CV analyses are insufficient to separate Models 2 and 9. Therefore, the modified stopping criterion saves considerable computing time here.

3.5 Comparing Statistical Methods or Explanatory Variable Sets

Recall that Section 3.1 described 80 statistical methods / descriptor set modeling strategies compared by ChemModlab. When comparing qualitatively different statistical methods and/or explanatory variable sets, there are two possible search implementations.

- *Tune then compare:*

Step 1 Tune each modeling strategy independently by repeating one of the algorithms in Section 3.4. For ChemModLab, this would mean 80 tuning searches.

Step 2 Compare the tuned models, again by applying one of the algorithms in Section 3.4.

R code for this implementation using algorithm 3 is given in Appendix A.

As we shall illustrate, the CV analyses in Step 1 can be re-used in Step 2, possibly leading to minimal further computing at Step 2. This approach is preferred when one wants to assess the performance of every modeling strategy after tuning. It requires many searches in Step 1, however.

- *Simultaneously tune and compare:* Carry out one search, simultaneously tuning and comparing the model strategies.

  This approach, we shall see, can require much less computing. Its drawback, however, is that it does not necessarily provide accurate estimation of the predictive performances of sub-optimal strategies; we just infer they are dominated by the winning strategy.

For simplicity, we will illustrate these two search implementations by comparing two statistical methods / descriptor sets for the AID362 Pub-

Chem assay. Extension to all 80 strategies explored by ChemModLab is straightforward. One strategy is a neural network with Burden number descriptors, which we call NN/Burden. NN/Burden was investigated in Section 3.4, and we already know that size=5 and decay=0.01 provides good values of the tuning parameters. The second strategy is k-nearest neighbors with Carhart atom pairs as explanatory variables, which we call KNN/Carhart.

For KNN/Carhart, we need to tune k, the number of the neighbors. We consider k in the range $1, 2, \ldots, 10$. Figure 3.6 shows the results of running Algorithms 2 and 3 in Sections 3.4.2 and 3.4.3. Algorithm 2 stops after 11 CV data splits and analyses, and the model with $k = 8$ emerges as the winner. (This agrees with more exhaustive computations to check our algorithm.) If we use the stopping criterion $p_0 = 1$ in (3.6), the algorithm stops after just eight data splits. With $p_0 = 2$, only five data splits are required. All these variants point to $k = 8$.

We now consider the tune-then-compare implementation for comparing NN/Burden with KNN/Carhart. For definiteness, we take $p_0 = 2$ in (3.6) as the stopping criterion. In Step 1, the two strategies are tuned independently, which has already been described. In Step 2, NN/Burden (size=5 and decay=0.01) is compared with KNN/Carhart ($k = 8$). Running Algorithm 3 in Section 3.4.3 for three data splits is sufficient to establish that tuned KNN/Carhart is better than tuned NN/Burden at significance level 0.05. The 95% confidence interval for the difference in mean h_{300} is $[2.31, 11.29]$. In Step 1, the total number of 10-fold CV analyses is $9 + 6 \times 4 + 5 \times 4 = 53$ for NN/Burden and $10 + 7 + 4 \times 3 = 29$ for KNN/Carhart. The same data splits were used for NN/Burden and KNN/Carhart. Thus, for Step 2, the first three CV analyses from Step 1 can be re-used and no further CV computations are required. Therefore, the total number of the executions of 10-fold CV in both steps to establish that KNN/Carhart (with $k = 8$) is superior is $53 + 29 = 82$.

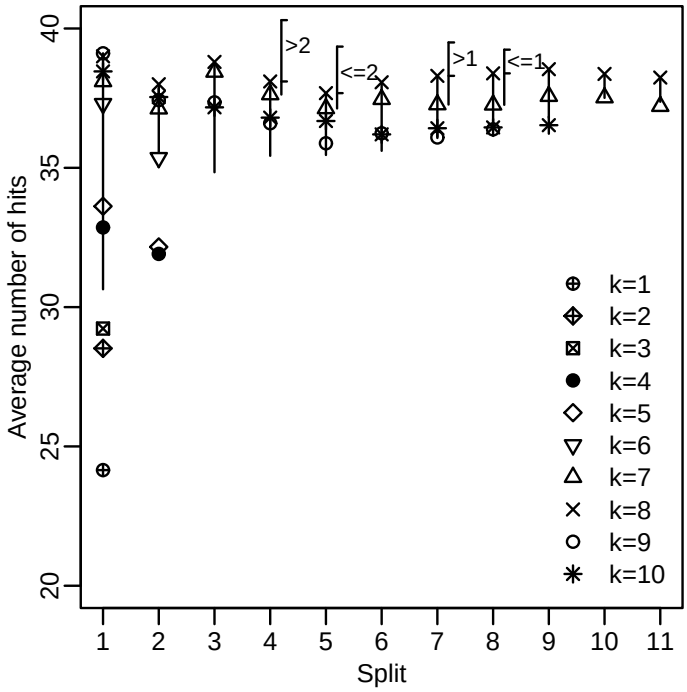

Figure 3.6: Data-adaptive algorithms applied to tuning k for k-nearest neighbors and Carhart atom pairs (KNN/Carhart) for the PubChem AID362 assay data.

For the simultaneous tune and compare implementation, the nine NN/Burden models (with different values of size and decay) and the 10 KNN/Carhart models (with different values of k) are put together as $m = 19$ initial models. The results of running Algorithm 3 in Section 3.4.3 with $p_0 = 2$ are shown in Table 3.3. We see after just one split, with the active compounds as blocks, three NN/Burden models and one KNN/Carhart model are eliminated. After two data splits, with splits as blocks, only one NN/Burden model and five KNN/Carhart models survive. After three splits, the remaining NN/Burden model is eliminated. The five KNN/Carhart models left survive through five splits, when the stopping criterion is satisfied. These models have $k = 6, 7, 8, 9$, and 10 and average h_{300} values of 35.6, 37.1, 37.7,

35.9, and 36.7, respectively. Therefore, we will again choose KNN/Carhart with $k = 8$ as the overall best model. The total number of CV executions is $19 + 15 + 6 + 5 \times 3 = 55$.

Number	NN/Burden model									KNN/Carhart model									
of splits	1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9	10
0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
1		✓	✓		✓	✓		✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
2		✓													✓	✓	✓	✓	✓
3															✓	✓	✓	✓	✓
4															✓	✓	✓	✓	✓
5															✓	✓	✓	✓	✓

Table 3.3: Simultaneously tuning NN/Burden and KNN/Carhart models for the PubChem AID362 assay data. The models surviving after each split are denoted by a check mark.

The second approach, simultaneously tuning and comparing models, requires less computer time here because the best KNN/Carhart models outperform all the NN/Burden models, and the latter can be quickly eliminated. In contrast, the tune-then-compare implementation spends much computational effort in tuning the inferior modeling strategy, NN/Burden. It does, however, lead to an accurate, quantitative assessment of the difference in predictive performance between NN/Nurden and KNN/Carhart.

3.6 Conclusions and Discussions

Throughout we used 10-fold CV, even for k-nearest neighbors where it is computationally straightforward to use n-fold (leave-one-out) CV. We did this for consistency across modeling methods: n-fold CV would be computationally infeasible for the method of neural networks also considered here and for many other methods. In addition, n-fold CV has well-known limitations. Theoretically, Shao (1993) showed its inconsistency in model selection. For applications like the molecular databases in PubChem, it also well known that n-fold CV can give over-optimistic estimates of predictive performance if the data have sets of similar compounds ("twins"). It is easy to predict one such compound's assay value from its near-analogs.

We illustrated that tuning a model may have a large effect on predictive performance. We also showed that the variation in CV performance estimates from one data split to another may necessitate multiple data splits for reliable comparison of different sets of tuning parameter values or of different tuned statistical modeling methods / explanatory variable sets. The data-adaptive algorithms developed in Sections 3.4 and 3.5 attempt to make reliable comparisons based on enough data splits, but sequentially focus the computational effort on models with better predictive performance.

The basic sequential algorithm in Section 3.4.1 uses data splits as a blocking factor, and hence requires at least two data splits for each candidate model. The variation in Section 3.4.2 uses individual observations as

the blocking factor, and can sometimes eliminate very inferior models after just one data split and CV analysis. To use observations as blocks, the performance measure must be an average over observations. The specialized h_{300} measure appropriate for the PubChem data set used throughout is of this type, as are more traditional metrics such as mean squared prediction error in regression problems or misclassification rate for classification problems.

The same approach can be applied to tuning a modeling strategy with respect to user-specified parameters and to comparing tuned modeling strategies. Simultaneously tuning and comparing appears to be highly efficient if there are poor modeling strategies that are dominated by other methods.

The proposed data-adaptive CV algorithm is sequential. At each iteration, a multiplicity-adjusted statistical test is developed to eliminate all inferior modeling strategies. An issue not addressed in this article is how to take account of the multiple testing across iterations. This is the topic of future study.

We gave an example where two different sets of explanatory variables were compared. In practice, some statistical models also have to be "tuned" with respect to selection of variables *within* a given set. This could also be done via our sequential CV algorithms, at least for a small number of candidate subsets of variables. Much adaptation would be necessary if there is a combinatorial explosion of possible subsets, and again this is future work.

In practice, some modeling parameters are usually treated as continuous factors, e.g., decay for NN. In the future, we would consider how to implement our sequential CV algorithms with continuous factors.

Bibliography

[1] Burden, F.R. (1989). Molecular identification number for substructure searches. *Journal of Chemical Information and Computer Sciences*, 36, 225–227.

[2] Burman, P. (1989). A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods. *Biometrika*, 76, 503–514.

[3] Dean, A. and Voss, D. (1999). *Design and Analysis of Experiments*. Springer.

[4] Dietterich, T.G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10, 1895–1923.

[5] Dudoit, S. and van der Laan, M.J. (2004). Asymptotics of cross-validated risk estimation in estimator selection and performance assessment. Technical Report 126, Division of Biostatististics, University of California, Berkeley. URL `www.bepress.com/ucbbiostat/paper126`.

[6] Hawkins, D.M., Basak, S.C., Mills, D. (2003). Assessing model fit by cross-validation. *Journal of Chemical Information and Computer Sciences*, 43, 579–586.

[7] Hughes-Oliver, J. M., Brooks, A. D., Welch, W. J., Khaledi, M. G.,Hawkins, D., Young, S. S., Patil, K., Howell, G. W., Ng, R. T., and Chu, M. T. (2007). ChemModLab: A web-based cheminformatics modeling laboratory. In review.

[8] Li, K.-C. (1987). Asymtotic optimality for c_p, c_l, cross-validation and generalized cross-validation: Discrete index set. *Annals of Statistics*, 15, 958–975.

[9] Maron, O. and Moore, A. W. (1997). The racing algorithm: model selection for lazy learners. *Artificial Intelligence Review*, 11: 193–225.

[10] Montgomery, D.C. (1997). *Design and Analysis of Experiments*, 5th ed. Wiley. Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge, Cambridge University Press.

[11] Shao, J. (1993). Linear model selection by cross-validation. *Journal of the American Statistical Association*, 88, 422, 486–494.

[12] Sinisi, S.E. and van der Laan, M.J. (2004). Loss-based cross-validated deletion/substitution/addition algorithms in estimation. Technical Report 143, Division of Biostatististics, University of California, Berkeley. URL `www.bepress.com/ucbbiostat/paper143`.

[13] Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, Series B, 36, 111–147.

[14] Stone, M. (1977). Asymptotics for and against cross-validation. *Biometrika*, 64, 29–35.

[15] Wang, Y. (2005). Statistical Methods for High Throughput Screening Drug Discovery Data. PhD Thesis, Department of Statistics and Actuarial Science, University of Waterloo.

[16] Yang, Y. H. (2006). Comparing learning methods for classification. *Statistica Sinica*, 16, 635–657.

[17] Zhang, P. (1993). Model selection via multifold validation. *The Annals of Statistics*, 21, 299–313.

Chapter 4

Correcting Cross Validation Bias for Linear Models

4.1 Introduction

The method of v-fold cross validation (CV) goes back at least to Mosteller and Tukey (1977), Breiman, Friedman, Olshen, and Stone (1984). It is widely used to assess the prediction error of statistical models and hence for selecting among candidate models.

In v-fold CV, a data set of n observations is split into v roughly equal-sized, mutually exclusive and exhaustive groups or *folds*. For $j = 1, \ldots, v$, fold j is held out as a test data set, the remaining $v - 1$ folds are used to fit the model, and the predictive error is computed for the observations in fold j. After looping through the v folds in this way, for a continuous response variable the n prediction errors are often summarized as a cross validated mean squared error (CVMSE). This serves as an estimate of the predictive mean squared error (MSE) that would arise if the model fit from all the data is used to predict the response variable in the future.

Although CV is widely used, there is little theory to guide the choice of v. Often $v = 10$ is chosen, but this is largely for computational reasons as CV requires v model fits in general. We study the impact of v on CVMSE

A version of this chapter will be submitted for publication. Authors: Shen H. and Welch W. J.

as an estimator of MSE for future observations in the case of linear models. Although n-fold (leave-one-out) CV is computationally straightforward for linear models, the lessons learned here will hopefully shed light on the behavior of CV for other modelling methods, where fairly small values of v are a computational necessity.

Suppose we have n observations on a response variable Y generated according to the linear model

$$\mathbf{Y} = \mathbf{X}^*\boldsymbol{\beta}^* + \boldsymbol{\epsilon}, \tag{4.1}$$

where $\mathbf{X}^*$ is an $n \times p^*$ matrix with $p^* < n$, and the random errors $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)^T$ are independent and identically distributed random variables with mean zero and variance σ^2. Note that the explanatory variables forming the columns of $\mathbf{X}^*$ are not necessarily known, hence the model selection problem. Throughout, a "*" superscript will denote quantities in the correct model that generated the data, as opposed to candidate models that are fit. In addition, we assume that all elements of $\boldsymbol{\beta}^*$ are non-zero, so that underfitting and overfitting are well defined in Section 4.2. In practice, the fitted model may differ from (4.1), and it is impossible to compute MSE directly. Hence, CVMSE is widely used instead of MSE. The definitions of underfitting, overfitting, MSE, and CVMSE are made more precise in Section 4.2.

There has been much theoretical study of CV. Efron (1986), Li (1987), and Stone (1974, 1977) focussed mainly on properties of n-fold (leave-one-out) CV. Shao (1993) and Zhang (1993) investigated asymptotic properties of various CV procedures for selecting among linear models. Burman (1989) established theoretical results for v-fold CV for general models. Dudoit and van der Laan (2003) derived asymptotic properties of a broad definition of CV (e.g., leave-one-out, v-fold, Monte Carlo, etc.) for model selection and performance assessment. Efron (2004) explored the connection between cross-validation and some penalty methods in terms of predictive error. van

der Laan, Dudoit and van der Vaart (2006) proposed a cross-validated adaptive epsilon-net estimator and proved its oracle inequality. van der Vaart, Dudoit and van der Laan (2006) considered choosing an estimator or model from a given class by cross validation and presented finite sample oracle inequalities. The original technical report for above two papers can be found in van der Laan and Dudoit (2003). Our article is most related to the work of Burman (1989) and Denby, Landwehr, and Mallows (2002), however.

Burman (1989) compared the performance of various CV methods, including the v-fold CV we study in this article. Because the results are for an arbitrary model, they are approximate. He also proposed correcting CVMSE to reduce its bias in estimating MSE in this general setting. The effectiveness of the correction depends on the true model, and the distribution of the original data, however, which are usually unknown. Moreover, as we illustrate in Sections 4.4 and 4.5, when the number of variables in the fitted model, p, approaches n, the correction sometimes does not help very much to reduce the considerable bias of CVMSE. In contrast, the correction we develop for CVMSE in the specific context of linear models does not depend on knowledge of the true model or on the distribution of the response variable. In particular, our correction is much larger when p/n is not small, and the need for correction is greatest.

Denby et al. (2002) used n-fold CV to estimate MSE and obtained the bias of CVMSE for linear regression. Their conclusions are similar to those of Efron (1986). They also suggested several modifications of CV, one of which is a special case of the bias correction we develop in Section 4.3.3. Possibly because their results are restricted to n-fold CV, they did not find substantial improvements from the bias correction when the fitted model omits terms appearing in the true model (underfitting). In contrast, we find our bias correction may give a very practical improvement for v-fold CV, including n-fold CV.

The paper is organized as follows. In Section 4.2 we give some definitions

and notation for MSE and CVMSE. Section 4.3 develops the statistical properties of CVMSE for v-fold CV. In particular, an exact and computable expression for the expected value of CVMSE over random data splits leads to a bias correction. In Sections 4.4 and 4.5 we give simulated and real examples to illustrate and support our results. We then summarize our conclusions in Section 4.6. The proofs of the results are outlined in the Appendix.

4.2 Definitions and Notation

The true model in (4.1) will usually be unknown. Instead of using the p^* explanatory variables leading to the design matrix $\mathbf{X}^*$, we fit the linear model using $p < n$ variables in $\mathbf{X}$.

Suppose, first, that $p < p^*$ and that $\mathbf{X}^* = (\mathbf{X}, \mathbf{X}^u)$. Such an 'underfitted" model based on $\mathbf{X}$ omits the columns $\mathbf{X}^u$ in $\mathbf{X}^*$. The corresponding partition of $\boldsymbol{\beta}^*$ in (4.1) is $\boldsymbol{\beta}^* = (\boldsymbol{\beta}^T, \boldsymbol{\beta}^{uT})^T$. Alternatively, suppose $p \geq p^*$ and $\mathbf{X} = (\mathbf{X}^*, \mathbf{X}^o)$, whereupon the fitted model may have unnecessary variables and is "overfitted". It is convenient to deal with fitting the true model, i.e., with $p = p^*$ and $\mathbf{X}^o$ empty, as a special case, even though there is really no overfitting. In practice, both underfitting and overfitting could arise. But here, the two issues are treated separately just for convenience.

We fit the model with design matrix $\mathbf{X}$ by least squares, i.e., $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$. We want to predict n new observations,

$$\mathbf{Y}_{\text{new}} = \mathbf{X}^*\boldsymbol{\beta}^* + \boldsymbol{\epsilon}_{\text{new}},$$

given by the (unknown) true model in (4.1), where $\boldsymbol{\epsilon}_{\text{new}}$ has the same distribution as $\boldsymbol{\epsilon}$ in (4.1) and is independent of $\boldsymbol{\epsilon}$. The least squares predictions are $\hat{\mathbf{Y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$.

The predictive MSE of the fitted model is defined as

$$\text{MSE} = \frac{1}{n} E_{\mathbf{Y}_{\text{new}}, \hat{\mathbf{Y}}} ||\mathbf{Y}_{\text{new}} - \hat{\mathbf{Y}}||^2, \tag{4.2}$$

where $||\mathbf{a}-\mathbf{b}||^2$ denotes $\sum_{i=1}^{n}(a_i - b_i)^2$ for any two vectors $\mathbf{a} = (a_1, \ldots, a_n)^T$ and $\mathbf{b} = (b_1, \ldots, b_n)^T$. Note that in this definition, the expectation is with respect to $\mathbf{Y}_{\text{new}}$ and $\hat{\mathbf{Y}}$ (and hence $\mathbf{Y}$). The MSE is also an average over the n observations. Different forms for the MSE will be given in Section 4.3.1 depending on whether we are under- or overfitting. For now we note that MSE cannot be computed in practice, because it depends on knowing the true model.

To estimate MSE, we perform a v-fold CV. Let $\mathbf{s} = \{s_1, \ldots, s_v\}$ denote a random split of the data into v roughly equally sized, mutually exclusive and exhaustive subsets of $\{1, \ldots, n\}$, i.e., $s_1 \cup \ldots \cup s_v = \{1, \ldots, n\}$ and $s_i \cap s_j = \emptyset$ for all $i \neq j$. The number of observations in s_j is n_{s_j}. Let $\mathbf{X}_{s_j}$ denote the $n_{s_j} \times p$ matrix formed from the rows of $\mathbf{X}$ indexed by s_j, and let $\mathbf{X}_{-s_j}$ be the $(n - n_{s_j}) \times p$ matrix formed from the rows of $\mathbf{X}$ not indexed by s_j. Similarly, $\mathbf{Y}_{s_j}$ and $\mathbf{Y}_{-s_j}$ are derived from the rows of $\mathbf{Y}$ indexed by s_j and the complementary rows, respectively. For $j = 1, \ldots, v$, a least-squares fit using $\mathbf{X}_{-s_j}$ and $\mathbf{Y}_{-s_j}$ gives $\hat{\boldsymbol{\beta}}_{-s_j} = (\mathbf{X}_{-s_j}^T \mathbf{X}_{-s_j})^{-1} \mathbf{X}_{-s_j}^T \mathbf{Y}_{-s_j}$ and least-squares predictions $\hat{\mathbf{Y}}_{s_j}^{\text{cv}} = \mathbf{X}_{s_j} \hat{\boldsymbol{\beta}}_{-s_j}$ of $\mathbf{Y}_{s_j}$. The CV estimate of MSE for the split $\mathbf{s}$ is defined as

$$\text{CVMSE}_{\mathbf{s}} = \frac{1}{n} \sum_{j=1}^{v} ||\mathbf{Y}_{s_j} - \hat{\mathbf{Y}}_{s_j}^{\text{cv}}||^2. \tag{4.3}$$

4.3 Statistical Properties of CVMSE

4.3.1 Expectation and bias of CVMSE

Lemma 1. Assume the true model is (4.1), where $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)^T$ are independent and identically distributed random variables with mean zero

and variance σ^2. For an overfitted model with p parameters $\boldsymbol{\beta}$ and design matrix $\mathbf{X} = (\mathbf{X}^*, \mathbf{X}^o)$, and a given split $\mathbf{s}$, we have

$$E(\text{CVMSE}_\mathbf{s}) = (1 + \frac{p}{n})\sigma^2 + \frac{\sigma^2}{n} \sum_{j=1}^{v} \sum_{i=1}^{n_{s_j}} \frac{d_{s_j,i}^2}{1 - d_{s_j,i}}, \tag{4.4}$$

where $d_{s_j,i} \geq 0$ is the i-th eigenvalue of the symmetric and non-negative definite matrix

$$\mathbf{P}_{s_j} = \mathbf{X}_{s_j}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}_{s_j}^T. \tag{4.5}$$

The proof of Lemma 1 is given in Appendix B.1.1. Note that the expectation in (4.4) is with respect to $\mathbf{Y}$; the split $\mathbf{s}$ is fixed.

It is easily shown that, when we overfit (including fitting the true model) the true MSE of prediction (e.g., Davison, 1997, p. 291) is

$$\text{MSE} = (1 + \frac{p}{n})\sigma^2. \tag{4.6}$$

This result and Lemma 1 lead immediately to the CV bias of a fixed split when overfitting.

Theorem 1. Under the overfitting conditions of Lemma 1, the bias of $\text{CVMSE}_\mathbf{s}$ in estimating MSE for a given split $\mathbf{s}$ is

$$E(\text{CVMSE}_\mathbf{s}) - \text{MSE} = \frac{\sigma^2}{n} \sum_{j=1}^{v} \sum_{i=1}^{n_{s_j}} \frac{d_{s_j,i}^2}{1 - d_{s_j,i}}.$$

Remark 1: If the fitted model is the true model, i.e., $\mathbf{X} = \mathbf{X}^*$, then $d_{s_j,i}$ in Theorem 1 is $d_{s_j,i}^*$, the i-th eigenvalue of $\mathbf{P}_{s_j}^* = \mathbf{X}_{s_j}^*({\mathbf{X}^*}^T\mathbf{X}^*)^{-1}{\mathbf{X}_{s_j}^*}^T$.

Remark 2: For n-fold (leave-one-out) CV, $\mathbf{P}_{s_j}$ in (4.5) is the scalar $\mathbf{H}_{jj}$, where

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T \tag{4.7}$$

is the "hat" matrix. The single eigenvalue of $\mathbf{H}_{jj}$ is $\mathbf{H}_{jj}$ itself. Hence,

$$E(\text{CVMSE}) = (1+\frac{p}{n})\sigma^2 + \frac{\sigma^2}{n}\sum_{j=1}^{n}\frac{\mathbf{H}_{jj}^2}{1-\mathbf{H}_{jj}} = (1+\frac{p}{n})\sigma^2 + \frac{\sigma^2}{n}\sum_{j=1}^{n}\left(\frac{1}{1-\mathbf{H}_{jj}} - \mathbf{H}_{jj} - 1\right)$$
$$= \frac{\sigma^2}{n}\sum_{j=1}^{n}\frac{1}{1-\mathbf{H}_{jj}},$$

because $\sum_{j=1}^{n}\mathbf{H}_{jj} = \text{tr}(\mathbf{H}) = p$. As there is only one split possible under n-fold CV, we drop the subscript $\mathbf{s}$ on CVMSE. This result for n-fold CV was given by Denby et al. (2002). Thus, Lemma 1 is a generalization to v-fold CV of their result.

Lemma 2. Assume the true model is (4.1), where $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)^T$ are independent and identically distributed random variables with mean zero and variance σ^2. For an underfitted model that omits the terms $\mathbf{X}^u\boldsymbol{\beta}^u$, and a given split $\mathbf{s}$, we have

$$E(\text{CVMSE}_\mathbf{s}) = \frac{1}{n}\boldsymbol{\beta}^{u^T}(\sum_{j=1}^{v}\tilde{\mathbf{X}}_{s_j}^{u^T}\tilde{\mathbf{X}}_{s_j}^{u})\boldsymbol{\beta}^u + (1+\frac{p}{n})\sigma^2 + \frac{\sigma^2}{n}\sum_{j=1}^{v}\sum_{i=1}^{n_{s_j}}\frac{d_{s_j,i}^2}{1-d_{s_j,i}},$$

where $\tilde{\mathbf{X}}_{s_j}^u = \mathbf{X}_{s_j}^u - \mathbf{X}_{s_j}(\mathbf{X}_{-s_j}^T\mathbf{X}_{-s_j})^{-1}\mathbf{X}_{-s_j}^T\mathbf{X}_{-s_j}^u$.

The proof is given in Appendix B.1.2.

For an underfitted model, it is easy to show that

$$\text{MSE} = \frac{1}{n}\boldsymbol{\beta}^{u^T}\mathbf{X}^{u^T}(\mathbf{I}-\mathbf{H})\mathbf{X}^u\boldsymbol{\beta}^u + (1+\frac{p}{n})\sigma^2, \tag{4.8}$$

This result and Lemma 2 lead immediately to the CV bias of a fixed split when underfitting.

Theorem 2. Under the underfitting conditions of Lemma 2, the bias of $\text{CVMSE}_\mathbf{s}$ in estimating MSE for a given split $\mathbf{s}$ is

$$E(\text{CVMSE}_\mathbf{s}) - \text{MSE} = \frac{1}{n}\boldsymbol{\beta}^{u^T}\left[\sum_{j=1}^{v}\tilde{\mathbf{X}}_{s_j}^{u^T}\tilde{\mathbf{X}}_{s_j}^{u} - \mathbf{X}^{u^T}(\mathbf{I}-\mathbf{H})\mathbf{X}^u\right]\boldsymbol{\beta}^u + \frac{\sigma^2}{n}\sum_{j=1}^{v}\sum_{i=1}^{n_{s_j}}\frac{d_{s_j,i}^2}{1-d_{s_j,i}}.$$

Corollary 1. Under the underfitting conditions of Lemma 2, if $\mathbf{X}^T\mathbf{X}^u = \mathbf{0}$, i.e., $\mathbf{X}$ and $\mathbf{X}^u$ are orthogonal, the result in Theorem 2 can be written as

$$E(\text{CVMSE}_\mathbf{s})-\text{MSE} = \frac{1}{n}\boldsymbol{\beta}^{u^T}\left[\sum_{j=1}^{v}\mathbf{X}_{s_j}^{u^T}(2\mathbf{W}_{s_j}+\mathbf{W}_{s_j}^2)\mathbf{X}_{s_j}^u\right]\boldsymbol{\beta}^u+\frac{\sigma^2}{n}\sum_{j=1}^{v}\sum_{i=1}^{n_{s_j}}\frac{d_{s_j,i}^2}{1-d_{s_j,i}},$$

where $\mathbf{W}_{s_j}$ is the positive definite matrix $\mathbf{X}_{s_j}(\mathbf{X}_{-s_j}^T\mathbf{X}_{-s_j})^{-1}\mathbf{X}_{s_j}^T$.

The proof is straightforward. From $\mathbf{X}^T\mathbf{X}^u = \mathbf{0}$ it follows that $\mathbf{X}_{s_j}^T\mathbf{X}_{s_j}^u = -\mathbf{X}_{-s_j}^T\mathbf{X}_{-s_j}^u$ and $\mathbf{H}\mathbf{X}^u = 0$.

Lemma 3. In Theorems 1 and 2,

$$\frac{\sigma^2}{n}\sum_{j=1}^{v}\sum_{i=1}^{n_{s_j}}\frac{d_{s_j,i}^2}{1-d_{s_j,i}} \geq 0.$$

The proof is given in Appendix B.1.3.

From Lemma 3, we can see that for an overfitted model (including fitting the true model), the bias of $\text{CVMSE}_\mathbf{s}$ is non-negative. For an underfitted model with $\mathbf{X}^T\mathbf{X}^u = \mathbf{0}$, Corollary 1 shows the bias is also non-negative.

Theorems 1 and 2 are exact results dealing with eigenvalues, thus it is not obvious to see the relationship between the bias of CVMSE and v and p. The following theorem gives this relationship through an approximate result.

Theorem 3. For the true model and overfitted models, suppose (i) $\mathbf{X}^T\mathbf{X} = \mathbf{I}$ (whereupon $\mathbf{P}_{s_j}$ in (4.5) equals $\mathbf{X}_{s_j}\mathbf{X}_{s_j}^T$), (ii) each fold for v-fold CV has exactly n/v observations, (iii) the rank of $\mathbf{X}_{s_j}$ is $\min(n/v, p)$, and (iv) the positive eigenvalues of $\mathbf{P}_{s_1}, \ldots, \mathbf{P}_{s_v}$ are all the same. Then

$$E(\text{CVMSE}_\mathbf{s}) - \text{MSE} = \begin{cases} \dfrac{p}{n(v-1)}\sigma^2 & \text{if } \dfrac{p}{n} < \dfrac{1}{v} \\ \dfrac{p^2}{n(n-p)}\sigma^2 & \text{if } \dfrac{1}{v} \leq \dfrac{p}{n} \leq \dfrac{v-1}{v} \end{cases}$$

The proof is given in Appendix B.1.4.

We can transform to make $\mathbf{X}$ orthonormal, i.e., $\mathbf{X}^T\mathbf{X} = \mathbf{I}$, without changing MSE, so there is no loss of generality from this condition in Theorem 3. The same "balance", and hence equality of eigenvalues, will not necessarily hold for every $\mathbf{X}_{s_j}\mathbf{X}_{s_j}^T$, however, and this is where the result is only an approximation. The simulations in Section 4.4, where the balance conditions do not hold, show that Theorem 3 nonetheless provides a good guide to the behavior of the bias.

Theorem 3 says that the bias of $\text{CVMSE}_\mathbf{s}$ decreases with v and increases with p when $p/n < 1/v$. If p/n exceeds $1/v$, the bias increases dramatically as p approaches n. For example, when $p/n = 0.1$ and $v = 5$, the approximate bias is $p/[n(v-1)]\sigma^2 = 0.025\sigma^2$. When $p/n = 0.8$ and $v = 10$, however, the bias is approximately $p^2/[n(n-p)]\sigma^2 = 3.2\sigma^2$.

4.3.2 Variance of CVMSE

Here, we give an exact result for n-fold CV and an approximate result for v-fold CV ($v < n$).

It is well known (e.g., Montgomery and Peck, 1992, p. 173) that for any fitted model (true, overfitted, or underfitted), CVMSE for n-fold CV can be calculated via

$$\text{CVMSE} = \frac{1}{n}\mathbf{r}^T\mathbf{\Lambda}^2\mathbf{r}, \tag{4.9}$$

where $\mathbf{r} = (\mathbf{I} - \mathbf{H})\mathbf{Y}$ is the vector of least squares residuals, $\mathbf{H}$ is from (4.7), and $\mathbf{\Lambda}$ is the $n \times n$ diagonal matrix with $\mathbf{\Lambda}_{ii} = (1 - \mathbf{H}_{jj})^{-1}$ for $j = 1, \ldots, n$, which is a measure of the leverage of observation i.

Theorem 4. Assume the true model is (4.1), where $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)^T$ are independent and identically distributed normal random variables with mean zero and variance σ^2. For n-fold CV and an overfitted model,

$$\text{Var}(\text{CVMSE}) = \frac{2\sigma^4}{n^2}\text{tr}[\mathbf{\Lambda}^2(\mathbf{I} - \mathbf{H})]^2,$$

and for an underfitted model that omits the terms $\mathbf{X}^u\boldsymbol{\beta}^u$,

$$\text{Var}(\text{CVMSE}) = \frac{2\sigma^4}{n^2}\text{tr}[\boldsymbol{\Lambda}^2(\mathbf{I}-\mathbf{H})]^2+\frac{4\sigma^2}{n^2}\boldsymbol{\beta}^{u^T}\mathbf{X}^{u^T}(\mathbf{I}-\mathbf{H})\boldsymbol{\Lambda}^2(\mathbf{I}-\mathbf{H})\boldsymbol{\Lambda}^2(\mathbf{I}-\mathbf{H})\mathbf{X}^u\boldsymbol{\beta}^u.$$

The proof is given in Appendix B.1.5. Note that $\boldsymbol{\Lambda}_{ii} = 1/(1 - \mathbf{H}_{jj})$ appears in Var(CVMSE) to the fourth power, which is only partially offset by $(\mathbf{I} - \mathbf{H})$ appearing up to either two or three times. In other words, if there are high-leverage observations with $\mathbf{H}_{jj}$ near 1, as there must be as p approaches n, CVMSE has very high sampling variation. We next give an analogous result for a special case of v-fold CV.

Theorem 5. Assume the fitted model is the true model. If a split $\mathbf{s}$ satisfies $\mathbf{X}^{*T}_{s_j}\mathbf{X}^*_{s_j} = \mathbf{X}^{*T}\mathbf{X}^*/v$ for $j = 1, \ldots, v$, then

$$\text{CVMSE}_\mathbf{s} = \frac{1}{n}\boldsymbol{\epsilon}^T\mathbf{Q}\boldsymbol{\epsilon},$$

where $\mathbf{Q}$ is the $n \times n$ matrix

$$\mathbf{Q} = \begin{pmatrix} \mathbf{I} + \frac{v}{v-1}\mathbf{X}^*_{s_1}(\mathbf{X}^{*T}\mathbf{X}^*)^{-1}\mathbf{X}^{*T}_{s_1} & \cdots & -(\frac{v}{v-1})^2\mathbf{X}^*_{s_1}(\mathbf{X}^{*T}\mathbf{X}^*)^{-1}\mathbf{X}^{*T}_{s_v} \\ \vdots & \ddots & \vdots \\ -(\frac{v}{v-1})^2\mathbf{X}^*_{s_v}(\mathbf{X}^{*T}\mathbf{X}^*)^{-1}\mathbf{X}^{*T}_{s_1} & \cdots & \mathbf{I} + \frac{v}{v-1}\mathbf{X}^*_{s_v}(\mathbf{X}^{*T}\mathbf{X}^*)^{-1}\mathbf{X}^{*T}_{s_v} \end{pmatrix}.$$

Further assuming that $\epsilon_1, \ldots, \epsilon_n$ are independent and identically distributed normal random variables with mean zero and variance σ^2, we have

$$\text{Var}(\text{CVMSE}_\mathbf{s}) = \frac{2\sigma^4}{n^2}\left[n + \frac{3v^3 - 3v^2 + v}{(v-1)^3}p^*\right].$$

The proof is given in Appendix B.1.6. We can see that the variance of $\text{CVMSE}_\mathbf{s}$ decreases with v and increases with p^* under the assumptions.

Remark: For a designed experiment, it may be possible to construct a split

for which $\mathbf{X}_{s_j}^{*T}\mathbf{X}_{s_j}^{*} = \frac{1}{v}\mathbf{X}^{*T}\mathbf{X}^{*}$. For example, suppose that

$$\mathbf{X}^{*} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}^{T},$$

and $v = 2$. The split that places observations 1, 2, 5, and 6 into fold 1, and observations 3, 4, 7, and 8 into fold 2, clearly satisfies the condition.

4.3.3 Bias correction for CVMSE

Theorem 1 gives the bias of CVMSE for an overfitted model (including fitting the true model) and a specific data split as

$$E(\text{CVMSE}_{\mathbf{s}}) - \text{MSE} = \frac{\sigma^2}{n}\sum_{j=1}^{v}\sum_{i=1}^{n_{s_j}}\frac{d_{s_j,i}^2}{1-d_{s_j,i}}.$$

This quantity could be subtracted from the computed CVMSE to correct for bias but would require estimation of σ^2. Empirically, such a correction using the mean square for residual to estimate σ^2 seems to perform poorly.

A multiplicative adjustment avoids estimation of σ^2. From (4.6), MSE $= (1 + p/n)\sigma^2$, and hence

$$\frac{E(\text{CVMSE}_{\mathbf{s}})}{\text{MSE}} = 1 + \frac{1}{n+p}\sum_{j=1}^{v}\sum_{i=1}^{n_{s_j}}\frac{d_{s_j,i}^2}{1-d_{s_j,i}}.$$

This factor leads to the following bias-corrected version of $\text{CVMSE}_{\mathbf{s}}$:

$$\text{CVMSE}_{\mathbf{s}}^{*} = \left(1 + \frac{1}{n+p}\sum_{j=1}^{v}\sum_{i=1}^{n_{s_j}}\frac{d_{s_j,i}^2}{1-d_{s_j,i}}\right)^{-1}\text{CVMSE}_{\mathbf{s}}. \tag{4.10}$$

Note that this bias correction is exact, i.e., $E(\text{CVMSE}_{\mathbf{s}}^{*}) = \text{MSE}$, without estimating any further quantities, as $d_{s_j,i}$ can be computed exactly from $\mathbf{X}$ for a specific split. In addition, the correction does not depend on knowing

the true model.

For underfitted models, the first term on the right-hand side of the bias in Theorem 2 depends on the unknown $\boldsymbol{\beta}^u$ and cannot be eliminated. The second term leads to the bias adjustment in (4.10), which we apply universally. For underfitted models, therefore, some of the positive bias is not corrected. Rather than being a problem, this may have some practical advantage: Underfitted models will be further penalized in model selection.

We next give some examples that show that the bias correction in (4.10) can work well.

4.4 Simulated-data Example

In this section we will conduct a simulation study to compare uncorrected and bias-corrected versions of CVMSE for n-fold, 10-fold, and 5-fold CV. The models considered include underfitted, true, and overfitted models. In addition to the bias correction in (4.10), we will evaluate a bias adjustment due to Burman (1989):

$$\text{CVMSE}_{\mathbf{s}}^{\text{B}} = \text{CVMSE}_{\mathbf{s}} + \frac{1}{n}||Y - X\hat{\boldsymbol{\beta}}||^2 - \frac{1}{v}\sum_{j=1}^{v}\frac{1}{n}||Y - X\hat{\boldsymbol{\beta}}_{-s_j}||^2. \quad (4.11)$$

For the simulation study, we take $n = 80$ observations and fit the following three models:

- Model 1: $Y = 2(1 + x_1 + x_2 + \cdots + x_9) + \epsilon$, where $p = 10$.
- Model 2: $Y = 2(1 + x_1 + x_2 + \cdots + x_9 + x_1x_2 + \cdots + x_8x_9) + \epsilon$, where $p = 46$.
- Model 3: $Y = 2(1 + x_1 + x_2 + \cdots + x_9 + x_{10} + x_1x_2 + \cdots + x_1x_{10} + \cdots x_9x_{10}) + \epsilon$, where $p = 56$.

Model 2 is the true model used to generate all data sets in the simulation. Models 1 and 3 underfit and overfit, respectively. The values of the explana-

tory variables $x_1, \ldots, x_{10}$ are sampled once for all simulations, independently from a uniform distribution on $[0, 1]$, and $\epsilon_1, \ldots, \epsilon_{80}$ are independently sampled from a standard normal distribution. Using model 2, 100 realizations of $Y_1, \ldots, Y_{80}$ are generated, with $\epsilon_1, \ldots, \epsilon_{80}$ independent from realization to realization.

For each model, the true predictive MSE is calculated from (4.8) for model 1 or from (4.6) for models 2 and 3. For each data realization, we compute $\text{CVMSE}_{\mathbf{s}}$ in (4.3), $\text{CVMSE}^*_{\mathbf{s}}$ in (4.10), and $\text{CVMSE}^{\text{B}}_{\mathbf{s}}$ in (4.11) for n-fold, 10-fold, and 5-fold CV. The same 5-fold or 10-fold split generated for a realization is applied across all models. The sample averages and standard deviations of the CVMSE values across the 100 data realizations are reported in Table 4.1. These results are also presented graphically in Figure 4.1.

			Average CVMSE		
Model	MSE	v	$\text{CVMSE}_{\mathbf{s}}$	$\text{CVMSE}^*_{\mathbf{s}}$	$\text{CVMSE}^{\text{B}}_{\mathbf{s}}$
1	1.63	n	1.79 (0.31)	1.76 (0.30)	1.78 (0.31)
		10	1.81 (0.32)	1.75 (0.31)	1.78 (0.31)
		5	1.87 (0.37)	1.77 (0.35)	1.80 (0.35)
2	1.58	n	2.69 (0.79)	1.51 (0.44)	2.67 (0.78)
		10	3.31 (1.04)	1.53 (0.48)	3.08 (0.97)
		5	4.55 (1.62)	1.51 (0.53)	3.84 (1.34)
3	1.70	n	4.18 (1.38)	1.69 (0.56)	4.14 (1.36)
		10	5.84 (2.04)	1.68 (0.59)	5.36 (1.87)
		5	12.69 (6.36)	1.78 (0.86)	10.35 (5.14)

Table 4.1: True predictive MSE and averages over 100 data realizations of CVMSE and two bias-corrected versions of CVMSE, for the simulated example. The numbers in brackets are the sample standard deviations.

By comparing with the MSE values, the average CVMSE values in Table 4.1 and Figure 4.1 are all seen to be positively biased. This is in accord with Lemma 3 and Corollary 1. Furthermore, the bias increases with p and decreases with v as suggested by Theorem 3. The magnitude of the bias for model 3 in particular is even larger than that suggested by Theorem 3,

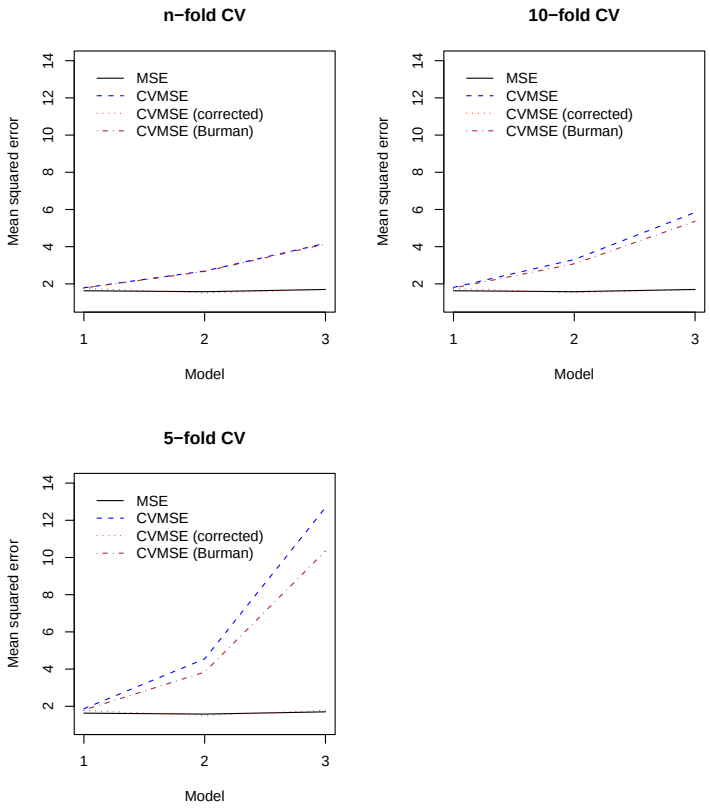

Figure 4.1: True predictive MSE and averages over 100 data realizations of CVMSE and two bias-corrected versions of CVMSE, for the simulated example.

because the "nice" balance conditions of the theorem do not hold here.

The sample standard deviation of CVMSE decreases with v and increases with p in Table 4.1. This is in accord with Theorem 5, which applies to model 2, though again the balance conditions of the theorem do not hold.

Thus, as an estimator of MSE, CVMSE has better bias and variance properties here as v increases. Even for n-fold CV, however, models 2 and 3 have such large biases that CVMSE is an extremely poor estimator of MSE.

Turning to the results after bias correction, we can see that neither $\text{CVMSE}^{*}_{\mathbf{s}}$ nor $\text{CVMSE}^{\text{B}}_{\mathbf{s}}$ help very much to reduce the bias of CVMSE for

the underfitted model 1. As noted in Section 4.3.3, the first term on the right-hand side of the bias in Theorem 2 cannot be eliminated, hence the too-small correction by $\text{CVMSE}^*_{\mathbf{s}}$. The effect of bias correction via $\text{CVMSE}^*_{\mathbf{s}}$ is very striking for models 2 and 3, however. The very substantial bias of CVMSE is virtually eliminated. In contrast, $\text{CVMSE}^{\text{B}}_{\mathbf{s}}$ leaves most of the bias uncorrected. We note that $\text{CVMSE}^*_{\mathbf{s}}$ has a smaller variance, too, compared with CVMSE or $\text{CVMSE}^{\text{B}}_{\mathbf{s}}$.

Bias correction also has important implications for model selection here. Model 2 has the smallest MSE and should therefore be chosen. Model 1, the underfitted model, has the smallest average CVMSE, however. Looking at the 100 individual realizations, if we select the "best" model based on the minimum standard CVMSE without correction, model 1 would be chosen for 91, 96 and 100 times out of the 100 data realizations, by n-fold, 10-fold and 5-fold CV, respectively. This is because model 2 has $p/n = 46/80$, which leads to a large positive bias. Alternatively, if the model selection is based on our $\text{CVMSE}^*_{\mathbf{s}}$ criterion, model 2 is chosen most often: for 58, 50, and 50 data realizations out of 100, for n-fold, 10-fold and 5-fold CV, respectively.

4.5 Biochemical-Process Example

We now present a real data example relating to a proprietary biochemical process to manufacture a medical device. The response variable is the thickness of a critical biologically active film, and there are 11 explanatory variables corresponding to process conditions. One hundred observations are available from running the plant under different process conditions.

We consider fitting 3 models. Model 1 has only an intercept term, i.e., $p = 1$. (Surprisingly, this model turns out to be a serious contender, hence its inclusion.) Model 2 has an intercept and all 11 explanatory variables entering linearly ($p = 12$). Model 3 has all the terms in model 2 plus all quadratic and bilinear interaction terms in $x_1, \ldots, x_{11}$. It has $p = 78$.

We compute $\text{CVMSE}_{\mathbf{s}}$, $\text{CVMSE}^*_{\mathbf{s}}$, and $\text{CVMSE}^{\text{B}}_{\mathbf{s}}$ for n-fold, 10-fold and

5-fold CV. To estimate the properties of these methods, we repeat the CV calculations for 10 independent 10-fold or 5-fold splits of the data. A split is used for all three models and for all three CVMSE measures, to block by split again. Table 4.2 presents the sample averages and standard deviations across the 10 splits for the CVMSE measures. Note that for model 3, CVMSE and corrected CVMSE values are available only for one of the 5-fold CV splits, due to numerical instability.

		Average CVMSE		
Model	v	$\text{CVMSE}_{\mathbf{s}}$	$\text{CVMSE}^{*}_{\mathbf{s}}$	$\text{CVMSE}^{\text{B}}_{\mathbf{s}}$
1	n	5.10	5.10	5.10
	10	5.13 (0.08)	5.13 (0.08)	5.12 (0.07)
	5	5.12 (0.06)	5.11 (0.06)	5.11 (0.05)
2	n	5.51	5.42	5.50
	10	5.55 (0.25)	5.38 (0.24)	5.48 (0.24)
	5	5.72 (0.31)	5.44 (0.29)	5.54 (0.27)
3	n	17.12	4.36	16.97
	10	40.79 (11.67)	5.30 (1.48)	37.01 (10.52)
	5	3245 (*)	10.58 (*)	2597 (*)

Table 4.2: Averages over 10 different data splits of CVMSE and two bias-corrected versions of CVMSE, for the biochemical-process example. The numbers in brackets are the sample standard deviations. For model 3 and $v = 5$ the CVMSE measures could be calculated for only one of the data splits.

For models 1 and 2, the averages in Table 4.2 are very similar across $\text{CVMSE}_{\mathbf{s}}$, $\text{CVMSE}^{*}_{\mathbf{s}}$, and $\text{CVMSE}^{\text{B}}_{\mathbf{s}}$ and across all three values of v, i.e., both bias corrections are very small. It is not surprising that the bias correction leading to $\text{CVMSE}^{*}_{\mathbf{s}}$ is small as p/n is only 0.01 or 0.11 for these two models. For model 3, where $p/n = 0.78$, however, there is a dramatic difference in the average $\text{CVMSE}_{\mathbf{s}}$ and $\text{CVMSE}^{*}_{\mathbf{s}}$ values, parallelling the results for the simulated data. We do not know the true MSE of the models for this real data set, but the implication is that $\text{CVMSE}_{\mathbf{s}}$ is very biased for model 3. Again, $\text{CVMSE}^{\text{B}}_{\mathbf{s}}$ has little bias correction.

If CV is used to choose a "best" model, then under n-fold CV, $\text{CVMSE}_{\mathbf{s}}$ and $\text{CVMSE}^{\text{B}}_{\mathbf{s}}$ would select model 1. For model 3, however, an enormous bias correction appears necessary, and $\text{CVMSE}^{*}_{\mathbf{s}}$ chooses this model. Under 10-fold CV, both $\text{CVMSE}_{\mathbf{s}}$ and $\text{CVMSE}^{\text{B}}_{\mathbf{s}}$ again choose model 1 for all 10 randomly generated splits, whereas $\text{CVMSE}^{*}_{\mathbf{s}}$ chooses model 1 four times and model 3 six times. Just like the simulated example in Section 4.4, this illustrates that standard CV can be very biased against complex linear models.

4.6 Conclusions and Discussion

We have investigated the bias and variance properties of CV for various fitted linear models. In particular, we have shown that there can be considerable bias, even for n-fold CV, if the number of parameters, p, is not small relative to n. The upward bias, i.e., overestimation of the magnitude of error, stems from using only a subset of the data to fit the model in CV.

All the theoretical results are for one *fixed* split, as in practice we want to know the properties of the CV analysis actually conducted. For a given data set and given split it is possible to compute an exact bias adjustment for overfitted models, including fitting the true model. For underfitted models that exclude important regression terms, some of the upward bias of CVMSE cannot be corrected by our results. We would argue that this only helps to select the "correct" model.

Under some balance conditions on the split, it is possible to show the impact of v and p on CV bias and variance. Both bias and variance decrease with v for overfitted models. A simulation study, where the balance conditions do not hold, showed qualitative agreement with these results.

Although underfitting and overfitting are treated separately in the paper, we have result analogous to Lemma 2 and Theorem 2 for combined condition. It is not reported here since the result is quite complicated.

For p/n approaching 1 for the fitted model, ridge regression is often used

as an alternative to least squares. The main purpose of implementing ridge regression, however, is to reduce the true predictive error rather than improve the estimation of the magnitude of error. Nonetheless, ridge regression may also lead to better CV properties, and this is worthy of investigation.

Cross validation predicts at the points in the explanatory variable space present in the data set at hand. Of course, of more practical interest is the predictive accuracy at new values, $\mathbf{x}_{\text{new}}$, of the explanatory variables. In this situation, as mentioned by Denby et al. (2002), the predictive MSE also depends on the difference between the expected value of $\mathbf{x}_{\text{new}}\mathbf{x}_{\text{new}}^T$ and $\mathbf{X}^T\mathbf{X}/n$ for the training data. The expectation and variance of CVMSE are unaffected by these considerations, and our results would also apply.

Minimum estimated predictive error is often used as the model selection criterion (Breiman, 1992; Shao, 1993; and Zhang, 1993). The considerable bias of CVMSE for complex linear models has important implications for model selection. In the biochemical application of Section 4.5, for example, the analysis performed at the time the experiment was conducted concluded that the null model with no explanatory variables could not be beaten. The same conclusion held after some reduction of the most complex linear model by backward elimination of terms. In other words, the process variables being manipulated appeared to have no predictive power according to CVMSE. This negative finding had serious implications for the future of an expensive project. In contrast, the bias-correction results of this paper suggest that a complex model may have predictive power for these data. The striking difference in these outcomes is due to the apparently enormous bias of CVMSE when fitting complex linear models.

Although the article is restricted to linear models, we speculate that CV would be similarly biased for other modelling strategies such as nonlinear regression, logistic regression, and neural networks. If our results do apply at least qualitatively, the implication is that CV is biased against more complex models even if they have superior true predictive performance. This is a

direction of future work.

It will be also interesting to see how our results on bias correction apply to variable selection.

Bibliography

[1] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*, Wadsworth and Brooks, Belmont.

[2] Breiman, L. (1992). Submodel Selection and Evaluation in Regression. The X-Random Case, *International Statistical Review* 60, 3. pp 291–319.

[3] Burman, P. (1989). A Comparative Study of Ordinary Cross-Validation, v-Fold Cross-Validation and the Repeated Learning-Testing Methods, *Biometrika* 76 503–514.

[4] Davison, A. C. (1997). *Bootstrap Methods and their Application*. Cambridge University Press.

[5] Denby, L., Landwehr, J. M., Mallows, C. L. (2002). Estimating predictive MSE by cross-validation, Technical report, available at `http://www.research.avayalabs.com/techabstractY.html\#ALR-2002-035`.

[6] Dudoit, S. and van der Laan, M.J. (2003). Asymptotics of Cross-Validated Risk Estimation in Estimator Selection and Performance Assessment, Technical Report 126, Division of Biostatististics, University of California, Berkeley. URL `www.bepress.com/ucbbiostat/paper126`.

[7] Efron, B. (1986). How Biased is the Apparent Error Rate of a Prediction Rule?, *Journal of the American Statistical Association* 81, 461–470.

[8] Efron, B. (2004). The Estimation of Prediction Error: Covariance Penalties and Cross-Validation, *Journal of the American Statistical Association* Vol. 99, No. 467,619–642.

[9] Li, K.-C. (1987). Asymtotic optimality for c_p, c_l, cross-validation and generalized cross-validation: Discrete index set, *Annals of Statistics* 15:958–975.

[10] Montgomery, D.C. and Peck, E.A. (1992). *Introduction to Linear Regression Analysis, 2nd edition*, Wiley.

[11] Mosteller, F. and Tukey, J. W. (1977). *Data Analysis and Regression.* Addison-Wesley Publishing Company.

[12] Searle, S.R. (1982). *Matrix Algebra Useful for Statistics*, Wiley.

[13] Shao, J. (1993). Linear Model Selection by Cross-Validation, *American Statistical Association* Vol. 88, No. 422, 486–494.

[14] Stone, M. (1974). Cross-Validatory Choice and Assessment of Statistical Predictions, *Journal of the Royal Statistical Society* Series B, 36, 111–147.

[15] Stone, M. (1977). Asymptotics for and against Cross-Validation, *Biometrika* 64(1): 29–35.

[16] van der Laan, M. J. and Dudoit, S. (2003). Unified Cross-Validation Methodology for Selection among Epsilon-Net Estimator: Finite Sample Oracle Inequalities and Examples, Technical Report 130, Division of Biostatististics, University of California, Berkeley. URL `www.bepress.com/ucbbiostat/paper130`.

[17] van der Laan, M. J. and Dudoit, S. (2006). The Cross-Validated Adaptive Epsilon-Net Estimator, *Statistics and Decisions* V. 24, No. 3, 373–395.

[18] van der Vaart, A. W., Dudoit, S. and van der Laan, M. J. (2006). Oracle Inequalities for Multi-Fold Cross-validation, *Statistics and Decisions* V. 24, No. 3, 351–371.

[19] Zhang, P. (1993). Model Selection Via Multifold Validation, *The Annals of Statistics* Vol. 21, No. 1, 299–313.

Chapter 5

Summary and Future Plan

5.1 Summary of the Thesis

In Chapter 2 of the thesis we review and compare three v-fold CV strategies: best single CV, repeated and averaged CV and double CV in terms of their predictive performance. The results are based on finite samples from a simulation and a real data example. Two modelling methods, NN and RF, are used. We find that repeated and averaged CV is the best strategy since it produces estimates of predictive performance with moderate bias and smallest variances. When enough data splits are made, the probability of choosing an optimal model (among the candidates available) is high. For single CV and repeated and averaged CV, we decompose the total bias to three sources: selection bias, sample size bias and sub-optimal model bias. This bias decomposition is used to explain why single CV biases, though considerable, partially cancel.

In Chapter 3 we illustrate that tuning a model may have a large effect on predictive performance. We also show that the variation in CV performance estimates from one data split to another may necessitate multiple data splits for reliable comparison of different sets of tuning parameter values or of different tuned statistical modeling methods / explanatory variable sets. We develop data-adaptive algorithms in an attempt to make reliable comparisons based on enough data splits, but sequentially focus the computational effort on models with better predictive performance. The same approach can be applied to tuning a modeling strategy with respect to user-specified parameters and to comparing tuned modeling strategies. Simultaneously

tuning and comparing appears to be highly efficient if there are poor modeling strategies that are dominated by other methods. Due to the variation of predictive performance between the CV splits, some poor models may outperform good models for certain splits of the data. The algorithm takes into account the randomness in CV via a multiplicity-adjusted statistical test so that the good models will not be eliminated and selection bias can also be avoided.

Moreover, the algorithms in Chapter 3 would be easy to implement for a parallel computer. And the R code in Appendix A is designed to facilitate adaptation to parallel computing.

In Chapter 4, for linear models, we show how CVMSE depends on the number of folds, v, used in cross validation, the number of observations, and the number of model parameters. We establish that the bias of CVMSE in estimating the true MSE decreases with v and increases with model complexity. In particular, the bias may be very substantial for models with many parameters relative to the number of observations, even if v is large. These results are used to correct CVMSE for its bias. We compare our proposed bias correction with that of Burman (1989), through simulated and real examples. We also illustrate that our method of correcting for the bias of CVMSE may change the results of model selection.

5.2 Future Plan

In Chapter 2, we show that repeated and averaged CV strategy outperforms best single CV and double CV based on simulation and real data examples. The performance measure we have used is AHR. We speculate that the conclusion can generalize to other performance measures such as misclassification rate, mean squared prediction error, AUC (area under ROC (receiver operating characteristic) curve) etc. Empirically, we could do more studies based on the above performance measures.

Furthermore, investigating theoretical properties of the three CV strate-

gies could be also a direction for the future research. To deal with the ranking measure like AHR or the number of hits found at certain number of compounds selected is very complicated and we may start with a simple performance measure such as mean squared prediction error for continuous response or misclassification rate for binary response.

In Chapter 3, we propose a sequential CV algorithm for model comparison. Currently, the algorithm is for data with binary response and the performance measure is the number of hits at certain number of compounds selected. We can easily generalize the algorithm to make it deal with continuous response and other performance measures with minor modifications. Recall that in this chapter we conduct Tukey's test based on the contribution of each active compound for one split of 10-fold CV and for more than one split of 10-fold CV, the test is based on the number of hits found at 300 compounds selected. Tukey's test based on the contribution of each active compound is only valid when the performance measure is the number of hits. For other measures, the adaptive test will be based on predictive performance obtained for all the random splits of 10-fold CV which have been run.

The data-adaptive CV algorithm is designed for eliminating poor models from all the candidate models. Thus it can be used for initial screening from a large amount of candidate models. The surviving models are "locally optimal" but not necessarily globally optimal. However, they may provide a direction for searching for globally optimal models. In the future, we could develop a new algorithm to search for the globally optimal models based on the result of initial screening.

The proposed data-adaptive CV algorithm is sequential. At each iteration, a multiplicity-adjusted statistical test is developed to eliminate all inferior modeling strategies. An issue not addressed in this chapter is how to take account of the multiple testing across iterations. This is the topic of future study.

We gave an example where two different sets of explanatory variables were compared. In practice, some statistical models also have to be "tuned" with respect to selection of variables *within* a given set. This could also be done via our sequential CV algorithms, at least for a small number of candidate subsets of variables. Much adaptation would be necessary if there is a combinatorial explosion of possible subsets, and again this is future work.

In Chapter 4, we show that for linear regression, the bias of CVMSE based on v-fold CV could be substantial when the number of parameters, p, in the model is big relative to the number of observations, n. Further exploration whether this finding can be generalized to some specific non-linear models and for drug discovery data would be an interesting research topic. We may start with conducting some simulations based on logistic regression and if we can find some interesting results we would do further theoretic work. However, the techniques used in linear regression may not be applicable to generalized linear model or other non-linear models. We need to consider appropriate tools or methods for solving the problem.

Appendix A

Appendix to Chapter 3

In this appendix, we provide R code for the adaptive algorithm in Chapter 3, Section 3.5 and give an example to illustrate how to implement the algorithm.

A.1 R Code for the Adaptive Algorithm

```
# This file contains all the functions to implement adaptive
# Tukey's test.
# The assessment measure is the number of hits at certain number
# of compounds selected.
# Currently the available methods are NN (neural network),
# RF (random forest), KNN (k nearest neighbor) and
# SVM (support vector machine)
#####################################################################
# Function adaptive.cv: implement data-adaptive algorithm based
# on v-fold CV to tune a given method with respect to all tuning
# parameters
#####################################################################
adaptive.cv<-function(y, x, v, para.frame, n.compounds=300,
method, alpha=0.05, max.split=100, stop.lowerbound=2)
# Arguments:
# y- response vector (0/1) for the entire data
# x- explanatory variable matrix for the entire data
# v- number of CV folds
```

```
# para.frame- parameter frame generated for a specific modeling
# method, i.e.,
# nn.parameter/rf.parameter/svm.parameter/knn.parameter function
# n.compounds- number of the compounds selected, default number
# is 300
# method- method used, can choose NN, RF, KNN or SVM
# alpha- significance level for Tukey's test, default is 0.05
# max.split- maximum number of splits for the adaptive algorithm,
# default number is 100
# stop.lowerbound- lower bound value to stop the adaptive algorithm,
# default value is 2
# Return values is a list with components:
# model- a dataframe of candidate models giving all tuning parameter
# combinations
# n.splits- number of splits execute when implementing the adaptive
# CV algorithm
# model.left- the surviving models for each iteration of the
# algorithm
# model.left.end- the surviving models when the algorithm stops
# nhit.ave- the average number of hits for the surviving models when
# the algorithm stops
# ie.ave- the average IE (initial enhancement) values for the surviving
# models when the algorithm stops
# lower.bound- the value of lower bound for each iteration of the
# algorithm
# nhit.matrix- matrix of the number of hits for each split of CV for
# the surviving models
# tukey- tukey value for each iteration of the algorithm
{
   n.y<-length(y)
```

```
n.x<-ncol(x)
n.active<-sum(y) # Number of y=1 cases
n.model<-nrow(para.frame)
# Fractional contributions to number of hits scores
frac.matrix<-matrix(0,n.model,n.active)
index.initial<-c(1:n.model) #Initial models
model.index.after<-NULL # Surviving models
# Initial Tukey value;
# Each component is Tukey value for each iteration
tukey<-NULL
# Initial lower bound based on fractional contribution to each
# active case
frac.lower.bound<-NULL
# Initial lower bound based on the number of hits
lower.bound<-NULL
# First split ("blocking"=active compounds)
sp<-1
# Call function get.phat.cv to compute predictive p hat for
# split 1
phat.1<-get.phat.cv(y,x,v,para.frame,method,sp)
# Call function get.nhit.all to compute the number of hits
# using phat.1
nhit.1<-get.nhit.all(y,phat.1,n.compounds)
# Compute fractional contributions for all the candidate models
for(i in 1:n.model)
{
   frac.matrix[i,]<-hit.fraction(phat.1[i,],y,n.compounds)
}
# Call function tukey.removal to remove poor models
# For the first split, the algorithm is based on fractional
```

```
# contribution to each active case
frac.tukey.test<-tukey.removal(frac.matrix,index.initial,alpha)
# frac.matrix.after<-frac.tukey.test$matrix.after
# Extract surviving models after 1 split of v-fold CV
model.index.after[[sp]]<-frac.tukey.test$model.index
# Transform Tukey value for 1 split to the number of hits
tukey[sp]<-frac.tukey.test$tukey*n.active
# Extract lower bound for split 1 based on the fractional
# contributions
frac.lower.bound[sp]<-frac.tukey.test$lower.bound
n.model.left<-length(model.index.after[[sp]])
nhit.1<-as.matrix(nhit.1)
# Form a new matrix which contains the number of hits for the
# surviving models
nhit.matrix<-as.matrix(nhit.1[model.index.after[[sp]],])
# Transform lower bound for split 1 to the number of hits
lower.bound[1]<-frac.lower.bound[1]*n.active
# If there is only one model left after 1 split of CV, the
# algorithm stops and returns some values
if(n.model.left==1)
{
   test.end.sp<-sp
   test.end.nhit<-nhit.1[model.index.after[[sp]]]
   test.end.ie<-test.end.nhit/sele.compound/n.active*n.y
   model.left<-model.index.after
   model.left.end<-model.index.after[[sp]]
   nhit.matrix.end<-test.end.nhit
}
# From split 2 on, "blocks"=splits
else
```

```
{
   sp<-2
   # Obtain tuning parameters for the surviving models
   para.frame.left<-as.matrix(para.frame[model.index.after
   [[sp-1]],])
   # Call function get.phat.cv to obtain predictive p hat for
   # split 2
   phat.temp<-get.phat.cv(y,x,v,para.frame.left,method,sp)
   # Call function get.nhit.all to obtain the number of hits for
   # split 2
   nhit.temp<-get.nhit.all(y,phat.temp,n.compounds)
   # Combine the number of hits for the two splits for all the
   # surviving models as a matrix
   nhit.matrix<-cbind(nhit.matrix,nhit.temp)
   # Call function tukey.removal to remove poor models
   nhit.tukey.test<-tukey.removal(nhit.matrix,
   model.index.after[[sp-1]], alpha)
   # Extract matrix of the number of hits for surviving models
   # after 2 splits
   nhit.matrix.after<-nhit.tukey.test$matrix.after
   # Extract index for the surviving models after 2 splits
   model.index.after[[sp]]<-nhit.tukey.test$model.index
   # Extract Tukey values for split 2
   tukey[sp]<-nhit.tukey.test$tukey
   # Extract lower bound for split 2
   lower.bound[sp]<-nhit.tukey.test$lower.bound
   n.model.left<-length(model.index.after[[sp]])
   # Keep removing poor models until the stopping criterion is
   # satisfied
   while(n.model.left>1 & sp<max.split &
```

```
   abs(lower.bound[sp])>stop.lowerbound)
   {
      sp<-sp+1
      cat("sp=",sp,"\n")
      para.frame.left<-as.matrix(para.frame[model.index.after
      [[sp-1]],])
      phat.temp<-get.phat.cv(y,x,v,para.frame.left,method,sp)
      nhit.temp<-get.nhit.all(y,phat.temp,n.compounds)
      nhit.matrix<-nhit.matrix.after
      nhit.matrix<-cbind(nhit.matrix,nhit.temp)
      nhit.tukey.test<-tukey.removal(nhit.matrix,
      model.index.after[[sp-1]], alpha)
      nhit.matrix.after<-nhit.tukey.test$matrix.after
      model.index.after[[sp]]<-nhit.tukey.test$model.index
      tukey[sp]<-nhit.tukey.test$tukey
      lower.bound[sp]<-nhit.tukey.test$lower.bound
      n.model.left<-length(model.index.after[[sp]])
   }
   test.end.sp<-sp
   if(n.model.left==1)
      test.end.nhit<-mean(nhit.matrix.after)
   else
      test.end.nhit<-apply(nhit.matrix.after,1,mean)
   test.end.ie<-test.end.nhit/sele.compound/n.active*n.y
   model.left<-model.index.after
   model.left.end<-model.index.after[[sp]]
   nhit.matrix.end<-nhit.matrix.after
   dimnames(nhit.matrix.end)<-NULL
}
return(list(model=para.frame,n.splits=test.end.sp,
```

```
   model.left=model.left, model.end=model.left.end,
   nhit.ave=test.end.nhit,ie.ave=test.end.ie,
   lower.bound=lower.bound,nhit.matrix=nhit.matrix.end,
   tukey=tukey))
}
#################################################################
# Function tukey.removal: remove poor modeling strategies based
# on Tukey's test for multiple comparison for RCBD (randomized
# complete block design)
#################################################################
tukey.removal<-function(nhit.matrix, model.index,alpha)
# Arguments:
# nhit.matrix- matrix for the number of hits for each split and
# method or matrix for the fraction of active compounds for each
# spit and each method
# model.index- the index number for each row of nhit.matrix in the
# original parameter list
# alpha- significance level
# Return values is a list with components:
# matrix.after- matrix of the number of hits for each split for
# the surviving models for one iteration of the algorithm
# model.index- index of the surviving models for one iteration of
# the algorithm
# tukey- tukey value for one iteration of the algorithm
# lower.bound- lower bound for one iteration of the algorithm
{
   n.row<-nrow(nhit.matrix) # Number of models
   n.col<-ncol(nhit.matrix) # Number of splits
   N<-n.row*n.col
   # Compute variation decomposition for RCBD
```

```
y.sum<-sum(nhit.matrix)
y.t.mean<-apply(nhit.matrix,1,mean)
y.t.sum<-y.t.mean*n.col
y.b.mean<-apply(nhit.matrix,2,mean)
y.b.sum<-y.b.mean*n.row
ss.total<-sum(nhit.matrix^2)-y.sum^2/N
ss.treat<-sum(y.t.sum^2)/n.col-y.sum^2/N
ss.block<-sum(y.b.sum^2)/n.row-y.sum^2/N
ss.error<-ss.total-ss.treat-ss.block
ms.error<-ss.error/(n.row-1)/(n.col-1)
# degree of freedom for Tukey's test
tukey.df<-(n.row-1)*(n.col-1)
if(tukey.df==1)
   tukey<-18.1
else
   tukey<-qtukey(1-alpha,n.row,tukey.df)*sqrt(ms.error/n.col)
nhit.order<-order(apply(nhit.matrix,1,mean),decreasing=T)
y.t.max<-y.t.mean[nhit.order[1]]
index.flag<-rep(0,n.row)
index.flag[nhit.order[1]]<-1
# Compare the best model to the other models, remove poor models
for(i in 2:n.row)
{
   y.t.i<-y.t.mean[nhit.order[i]]
   if((y.t.max-y.t.i)<=tukey) index.flag[nhit.order[i]]<-1
}
model.index.after<-model.index[index.flag==1]
nhit.matrix.after<-nhit.matrix[index.flag==1,]
nhit.matrix.after<-as.matrix(nhit.matrix.after)
# No need to compute the lower bound if there is only one model
```

```
   # left
   if(ncol(nhit.matrix.after)==1)
   {
      lower.ci<-NULL
   }
   # If there are more than two models left, compute the lower
   # bound
   else
   {
      y.t.mean.after<-apply(nhit.matrix.after,1,mean)
      y.t.max.after<-max(y.t.mean.after)
      i.max<-which(y.t.mean.after==y.t.max.after)[1]
      lower.ci<--max(abs(y.t.max.after-y.t.mean.after[-i.max]
      -tukey))
   }
   return(list(matrix.after=nhit.matrix.after,
   model.index=model.index.after, tukey=tukey,
   lower.bound=lower.ci))
}
#################################################################
# The following two functions nn.parameters and nn.run relate
# to neural networks.
# If you change nn.parameters, you will need to change the nnet
# calling sequence in nn.run.
#################################################################
#################################################################
# Function nn.parameters: produce a parameter frame for all the
# combinations of the tuning parameters "size" and "decay" for
# NN (neural network)
#################################################################
```

```
nn.parameters<-function(size,decay)
# Arguments:
# size- parameter vector for the tuning parameter size
# decay- parameter vector for the tuning parameter decay
# Return values:
# para.frame- a data frame for the values of tuning parameters
# for NN
{
   mylist.nn<-list(decay,size)
   para.frame<-expand.grid(mylist.nn)
   names(para.frame)<-c("decay","size")
   return(para.frame)
}
################################################################
# Function nn.run: compute predictive p hat based on one fold
# of v-fold CV for one combination of the NN parameters.
# When fit NN model, using 3 fits, choose the one with the
# smallest entropy
################################################################
nn.run<-function(y.train,x.train,x.test,para.vector)
# Arguments:
# y.train- response vector for the training data
# x.train- explanatory variable matrix for the training data
# x.test- explanatory variable matrix for the test data
# para.vector- one combination of parameter values
# (size and decay)
# Return values:
# phat.nn.predict- predictive p hat for CV based on NN
{
   library(nnet)
```

```
   y.train<-as.factor(y.train)
   data.nn<-cbind(y.train,x.train)
   nn.fit<-NULL
   nn.entropy<-NULL
   # Fit NN 3 times
   for(fit in 1:3)
   {
      nn.fit[[fit]]<-nnet(y.train~.,data=data.nn,
      decay=para.vector[1], size=para.vector[2],
      maxit=2000,abstol=0.001)
      nn.entropy[fit]<-nn.fit[[fit]]$value
   }
   # Pick the fit with the smallest entropy
   fit.best<-order(nn.entropy)[1]
   nn.fit.best<-nn.fit[[fit.best]]
   phat.nn.predict<-predict(nn.fit.best,newdata=x.test,
   type="raw")
   return(phat.nn.predict)
}
#####################################################################
# The following two functions rf.parameters and rf.run relate to
# random forest.
# If you change rf.parameters, you will need to change the
# randomForest calling sequence in rf.run.
#####################################################################
#####################################################################
# Function rf.parameters: produce a parameter frame for all the
# combinations of the tuning parameters "mtry" and "nodesize" for
# RF (random forest)
#####################################################################
```

```
rf.parameters<-function(mtry,nodesize)
# Arguments:
# mtry- parameter vector for the tuning parameter mtry
# nodesize- parameter vector for the tuning parameter nodesize
# Return values:
# para.frame- a data frame for the values of tuning parameters
# for RF
{
   mylist.rf<-list(nodesize,mtry)
   para.frame<-expand.grid(mylist.rf)
   names(para.frame)<-c("nodesize","mtry")
   return(para.frame)
}
################################################################
# Function rf.run: compute predictive p hat based on one fold
# of v-fold CV for one combination of the RF paramaters
################################################################
rf.run<-function(y.train,x.train,x.test,para.vector)
# Arguments:
# y.train- response vector for the training data
# x.train- explanatory variable matrix for the training data
# x.test- explanatory variable matrix for the test data
# para.vector- one combination of parameter values
# (mtry and nodesize)
# Return values:
# phat.rf.predict- predictive p hat for CV based on RF
{
   library(randomForest)
   y.train<-as.factor(y.train)
   rf.fit<-randomForest(y=y.train,x=x.train,ntree=100,
```

```
   notesize=para.vector[1], mtry=para.vector[2], importance=F)
   phat.rf.predict<-predict(rf.fit,x.test,type="prob")[,2]
   return(phat.rf.predict)
}
################################################################
# The following two functions svm.parameters and svm.run relate
# to support vector machine. If you change svm.parameters, you
# will need to change the svm calling sequence in svm.run.
################################################################
################################################################
# Function svm.parameters: produce a parameter frame for all
# the combinations of the tuning parameters "gamma" and "cost"
# for SVM (support vector machine)
################################################################
svm.parameters<-function(gamma,cost)
# Arguments:
# gamma- parameter vector for the tuning parameter gamma
# cost- parameter vector for the tuning parameter cost
# Return values:
# para.frame- a data frame for the values of tuning parameters
# for SVM
{
   mylist.svm<-list(gamma,cost)
   para.frame<-expand.grid(mylist.svm)
   names(para.frame)<-c("gamma","cost")
   return(para.frame)
}
################################################################
# Function svm.run: compute predictive p hat based on one fold
# of v-fold CV for one combination of the SVM parameters
```

```
#################################################################
svm.run<-function(y.train,x.train,x.test,para.vector)
# Arguments:
# y.train- response vector for the training data
# x.train- explanatory variable matrix for the training data
# x.test- explanatory variable matrix for the test data
# para.vector- one combination of parameter values
# (gamma and cost)
# Return values:
# phat.svm.predict- predictive p hat for CV based on SVM
{
   library(class)
   library(e1071)
   y.train<-as.factor(y.train)
   svm.fit<-svm(x=x.train,y=y.train,gamma=para.vector[1],
   cost=para.vector[2],
   probability=TRUE)
   svm.prob.obj<-predict(svm.fit,x.test,probability=TRUE)
   phat.svm.predict<-attr(svm.prob.obj,"probabilities")[,2]
   return(phat.svm.predict)
}
#################################################################
# The following two functions knn.parameters and knn.run relate
# to k-nearest neighbor. If you change knn.parameters, you will
# need to change the knn calling sequence in knn.run.
#################################################################
#################################################################
# Function knn.parameters: produce a parameter frame for all
# the values of the tuning parameter k for KNN
# (k-nearest neighbor)
```

```
#################################################################
knn.parameters<-function(knn.k)
# Arguments:
# knn.k- parameter vector for the tuning parameter k
# Return values:
# para.frame- a data frame for the values of tuning parameters
# for KNN
{
   para.frame<-as.matrix(knn.k)
   rownames(para.frame)<-c(1:length(knn.k))
   colnames(para.frame)<-"K"
   return(para.frame)
}
#################################################################
# Function knn.run: compute predictive p hat based on one fold
# of v-fold CV for one value of the KNN parameter (k)
#################################################################
knn.run<-function(y.train,x.train,x.test,para)
# Arguments:
# y.train- response vector for the training data
# x.train- explanatory variable matrix for the training data
# x.test- explanatory variable matrix for the test data
# para- one value of k
# Return values:
# phat.knn.predict- predictive p hat for CV based on KNN
{
   library(class)
   y.train<-as.factor(y.train)
   knn.obj<-knn(x.train,x.test,y.train,k=para,prob=TRUE)
   phat.knn.predict<-knn.prob(knn.obj,level=2)
```

```
   return(phat.knn.predict)
}
knn.prob<-function(knn.obj,level)
{
   prob<-attributes(knn.obj)$prob
   level.name<-levels(knn.obj)[level]
   prob[knn.obj!=level.name]<-1-prob[knn.obj!=level.name]
   return(prob)
}
#################################################################
# Function get.phat.cv: get predictive p hat of all the
# parameters in the parameter frame based on one split of
# v-fold CV for a given method.
# The available methods are: NN, RF, SVM and KNN
#################################################################
get.phat.cv<-function(y,x,v,para.frame,method,sp)
# Arguments:
# y- response vector for the entire data
# x- explanatory variable matrix for the entire data
# v- the number of the folds to conduct CV, e.g, 10-fold CV
# para.frame- parameter frame generated from nn.parameter
# /rf.parameter/svm.parameter/knn.parameter function
# method- a given method, e.g., "NN", "RF", "SVM", "KNN"
# sp- split number
# Return values:
# phat.predict- predictive p hat for one run of v-fold CV for
# a given method
{
   n<-length(y)
   para.frame<-as.matrix(para.frame)
```

```
   n.model<-nrow(para.frame)
   phat.predict<-matrix(0,n.model,n)
   set.seed(100*sp)
   foldvector<-sample(1:v,n,replace=T)
   for(i in 1:v)
   {
      y.train<-y[foldvector!=i]
      x.train<-x[foldvector!=i,]
      x.test<-x[foldvector==i,]
      for(j in 1:n.model)
      {
         if(method=="NN")
            phat.predict[j,foldvector==i]<-nn.run(y.train,
            x.train,x.test, para.frame[j,])
         if(method=="RF")
            phat.predict[j,foldvector==i]<-rf.run(y.train,
            x.train,x.test, para.frame[j,])
         if(method=="SVM")
            phat.predict[j,foldvector==i]<-svm.run(y.train,
            x.train,x.test, para.frame[j,])
         if(method=="KNN")
            phat.predict[j,foldvector==i]<-knn.run(y.train,
            x.train,x.test, para.frame[j,])
      }
   }
   return(phat.predict)
}
#################################################################
# Function get.nhit.all: get the number of hits at certain
# number of compounds selected for the phat matrix in a
```

```
# parameter list
#################################################################
get.nhit.all<-function(y,phat.predict,n.compounds)
# Arguments:
# y- response vector for the entire data
# phat.predict- predictive hat matrix obtained from get.phat.cv
# n.compounds- the number of the compounds selected, e.g., 300
# Return values:
# nhit- number of hits for each model
{
   n.model<-nrow(phat.predict)
   nhit<-rep(0,n.model)
   for(i in 1:n.model)
   {
      nhit[i]<-hit.curve(phat.predict[i,], y,n.compounds,
      plot.it=F)$nhitlast
   }
   return(nhit)
}
#####################################################################
# Function hit.curve: compute the number of hits at certain
# number of compounds selected and plot hitcurve. Adapted from
# William J. Welch's R code at
# http://hajek.stat.ubc.ca/~will/ddd/, under the title "Functions''
#####################################################################
hit.curve <- function(phat, y, n.compounds = 300, plot.it = T)
# Arguments:
# phat- predictive phat vector for each method based on one v-fold
# CV
# y- response vector for the entire data
```

```
# n.compounds- number of the compounds selected, the default number
# is 300
# plot.it- plot hit curve or not
# Return values is a list with components:
# select- number of compounds (cases) in each tied p hat group
# p- unique p hat values
# nhits- number of hits in each tied p hat group
# nhitlast- number of hits after n.compounds compounds selected
{
   # unique phats, sorted with largest first.
   uniq.phat<-rev(sort(unique(phat)))
   select<-vector("numeric", length(uniq.phat))
   nhits<-vector("numeric", length(uniq.phat))
   for (i in 1:length(uniq.phat))
   {
      cases.sel<-(phat == uniq.phat[i])
      select[i]<-sum(cases.sel)
      nhits[i]<-sum(y[cases.sel])
      if (sum(select[1:i])>=n.compounds || i==length(uniq.phat))
      {
         i.max<-i
         break
      }
   }
   uniq.phat<-uniq.phat[1:i.max]
   nhits<-nhits[1:i.max]
   select<-select[1:i.max]
   exp.hits<-uniq.phat * select
   # Plot hitcurve
   if (plot.it)
```

```
{
   # Just set up axes, etc.
   x.max<-sum(select)
   y.max<-max(sum(nhits), sum(exp.hits))
   plot(1, 1, type = "n", xlim=c(0,x.max), ylim=c(0,y.max),
   xlab = "Number of compounds selected",
   ylab = "Number of actual/expected hits")
   # Plot the cumulative number of hits.
   cum.select.last<-0
   cum.nhits.last<-0
   cum.exp.hits.last<-0
   for (i in 1:i.max)
   {
      cum.select<-cum.select.last+select[i]
      cum.nhits<-cum.nhits.last+nhits[i]
      cum.exp.hits<-cum.exp.hits.last+uniq.phat[i]*select[i]
      # Always plot points.
      points(cum.select, cum.nhits, pch="+",
      cex=par()$cex * 0.9)
      points(cum.select, cum.exp.hits, pch="O",
      cex=par()$cex * 0.7)
      # Join points with lines if gap is sufficient.
      if (select[i]/x.max > 0.02)
      {
         lines(c(cum.select.last+1, cum.select),
         c(cum.nhits.last+nhits[i]/select[i], cum.nhits),
         lty = 1)
         lines(c(cum.select.last+1, cum.select),
         c(cum.exp.hits.last+uniq.phat[i], cum.exp.hits),
         lty = 2)
```

```
      }
      cum.select.last<-cum.select
      cum.nhits.last<-cum.nhits
      cum.exp.hits.last<-cum.exp.hits
   }
   legend(0.05*n.compounds, 0.9*y.max, c("Actual hits",
   "Expected hits"), pch = "+O")
}
# Calculate number of hits after n.compounds cases selected
nhit.select<-0
select.cum<-cumsum(select)
nhits.cum<-cumsum(nhits)
for(i in 1:(length(uniq.phat)-1))
{
   if(select.cum[i]==n.compounds)
   {
      nhit.select<-nhits.cum[i]; break
   }
   if(select.cum[i+1]==n.compounds)
   {
      nhit.select<-nhits.cum[i+1]; break
   }
   if(n.compounds>select.cum[i] & n.compounds<select.cum[i+1])
   {
      nhit.select<-nhits.cum[i]+nhits[i+1]/select[i+1]*
      (n.compounds-select.cum[i])
      break
   }
}
return(list(select = select, p = uniq.phat, nhits = nhits,
```

```
   nhitlast=nhit.select))
}
################################################################
# Function hit.fraction: compute the fraction for each active
# cases (class 1) at n.compounds selected based on ordered
# predictive p hat
################################################################
hit.fraction<-function(phat, y, n.compounds)
# Arguments:
# phat- predictive p hat vector for some model based on v-fold
# CV
# y-response vector (numeric 0/1) for the entire data
# n.compounds- maximum number of compounds to be selected
# Return values:
# hit.sign.frac- fraction contribution to each active compound
{
   hit.which.y<-which(y==1)
   num.y.1<-sum(y==1)
   hit.sign.frac<-rep(0,num.y.1)
   uniq.phat <- sort(unique(phat),decreasing=T)
   select <- vector("numeric", length(uniq.phat))
   nhits  <- vector("numeric", length(uniq.phat))
   hit.which<-NULL
   for (i in 1:length(uniq.phat))
   {
      cases.sel<- (phat==uniq.phat[i])
      which.sel<-which(phat==uniq.phat[i])
      select[i]<- sum(cases.sel)
      nhits[i]<- sum(y[cases.sel])
      hit.which.temp<-which.sel[y[which.sel]==1]
```

```
      if (sum(select[1:i]) >= n.compounds || i==length(uniq.phat))
      {
         i.max <- i
         if(length(hit.which.temp)>0)
            hit.which.rest<-hit.which.temp
         else
            hit.which.rest<-"NA"
         break
      }
      else
         hit.which<-c(hit.which,hit.which.temp)
   }
   num.select.last<-n.compounds-sum(select[1:(i.max-1)])
   if(sum(select[1:i.max])> n.compounds)
       hit.rest.frac<-num.select.last/select[i.max]
   if(sum(select[1:i.max])== n.compounds)
       hit.rest.frac<-1
   for(i in 1:num.y.1)
   {
      if (any(hit.which==hit.which.y[i]))
         hit.sign.frac[i]<-1
      if (any(hit.which.rest==hit.which.y[i]))
         hit.sign.frac[i]<-hit.rest.frac
   }
   return(hit.sign.frac)
}
```

A.2 An Example to Implement the Adaptive CV Algorithm

We now use the NN/Burden number strategy (in Section 3.3 of Chapter 3) as an example to show how to implement the adaptive CV algorithm.

Suppose that burden.y is a vector containing binary response (0/1) and burden.x is a matrix containing 24 explanatory variables for the data set AID362, Burden numbers. The data set for AID362, Burden numbers can be downloaded from `http://eccr.stat.ncsu.edu/ChemModLab/ViewResults.aspx?id=2007-01-24-14-30-31/Split1`, burden.y can be obtained from the link "Responses" and burden.x can be obtained from the link "Descriptors". For the NN method, as in the chapter, the tuning parameters "size" and "decay" take values size=5, 7, 9 and decay=0.1, 0.01, 0.001. Suppose that 10-fold CV is carried out, the number of compounds selected is 300, the significance level is 5%, and the algorithm stopping criterion is:

only 1 model left or the number of splits ≥ 100 or |lower bound| < 2.

R code to implement the adaptive CV algorithm is given below. The user may get a slightly different result due to some randomness in the NN algorithm.

```
nn.size<-c(5,7,9)
nn.decay<-c(0.1,0.01,0.001)
# form a data frame for the 3*3 combinations of the tuning
# parameters
nn.list<-nn.parameters(nn.size,nn.decay)
v<-10
n.compounds<-300
alpha<-0.05
max.split<-100
stop.lowerbound<-2
```

```
# call function adative.cv
nn.burden.test<-adaptive.cv(burden.y, burden.x, v, nn.list,
n.compounds, "NN" ,alpha, max.split ,stop.lowerbound)
```

The output is as follows.

```
> nn.burden.test
$model - All the combinations of the tuning parameters
 decay size
1 0.100    5
2 0.010    5
3 0.001    5
4 0.100    7
5 0.010    7
6 0.001    7
7 0.100    9
8 0.010    9
9 0.001    9
$n.splits - The number of splits when the algorithm stops
[1]9
$model.left - The models left for each iteration
$model.left[[1]] - The models left after the first iteration
[1] 2 3 5 6 8 9
$model.left[[2]] - The models left after the second iteration
[1] 2 3 5 6 8 9
$model.left[[3]]
[1] 2 3 5 6 8 9
$model.left[[4]]
[1] 2 3 5 6 8 9
$model.left[[5]]
[1] 2 5 6 8 9
$model.left[[6]]
```

```
 [1] 2 5 6 8 9
 $model.left[[7]]
 [1] 2 5 6 8 9
 $model.left[[8]]
 [1] 2 5 6 8 9
 $model.left[[9]]
 [1] 2 5 6 8 9
 $model.end - The models left at the end
 [1] 2 5 6 8 9
 $nhit.ave - The average number of hits for the left models
             at the end
 [1] 31.88889 29.88889 29.55556 30.44444 31.00000
 $ie.ave - The average IE (initial enhancement) for the left
           models at the end
 [1] 7.573611 7.098611 7.019444 7.230556 7.362500
 $lower.bound - Lower bound for each iteration
 [1] -10.5876 -6.9604 -5.0824 -3.1646 -3.2685 -2.6037 -2.1405
     -2.1988 -1.6719
 $nhit.matrix - The number of hits for each split for the left
                models at the end
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]   34   32   30   31   27   34   35   29   35
 [2,]   32   28   31   28   27   31   30   31   31
 [3,]   29   28   33   29   27   33   30   29   28
 [4,]   33   30   30   30   25   31   35   31   29
 [5,]   27   31   32   30   31   33   33   31   31
 $tukey -Tukey's value for each iteration
[1] 11.5860 8.4604 6.0824 4.1646 3.8685 3.2704 2.9977 2.6988
    2.5608
```

Appendix B

Appendix to Chapter 4

B.1 Proofs

We assume throughout the existence of all required matrix inverses.

B.1.1 Proof of Lemma 1

For a given split $\mathbf{s}$, we have

$$E(\text{CVMSE}_{\mathbf{s}}) = E_{\mathbf{Y}}[\frac{1}{n}\sum_{j=1}^{v}||\mathbf{Y}_{s_j} - \hat{\mathbf{Y}}_{s_j}^{\text{cv}}||^2] = \frac{1}{n}\sum_{j=1}^{v}\sum_{i=1}^{n_{s_j}} E_{\mathbf{Y}}(Y_{s_j,i} - \hat{Y}_{s_j,i}^{\text{cv}})^2,$$

where $Y_{s_j,i}$ and $\hat{Y}_{s_j,i}^{\text{cv}}$ are element i of $\mathbf{Y}_{s_j}$ and $\hat{\mathbf{Y}}_{s_j}^{\text{cv}}$, respectively. We will compute $E(Y_{s_j,i} - \hat{Y}_{s_j,i}^{\text{cv}})^2$ by writing it as $E^2(Y_{s_j,i} - \hat{Y}_{s_j,i}^{\text{cv}}) + \text{Var}(Y_{s_j,i} - \hat{Y}_{s_j,i}^{\text{cv}})$.

To calculate $E(Y_{s_j,i} - \hat{Y}_{s_j,i}^{\text{cv}})$, write the parameters in the fitted model as $\boldsymbol{\beta}^T = ({\boldsymbol{\beta}^*}^T, {\boldsymbol{\beta}^o}^T)$, where the vector of overfitted parameters, $\boldsymbol{\beta}^o = (0, \ldots, 0)^T$, has $p - p^*$ elements. Therefore, $\mathbf{Y} = \mathbf{X}^*\boldsymbol{\beta}^* + \boldsymbol{\epsilon} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$. Similarly, $\mathbf{Y}_{s_j} = \mathbf{X}_{s_j}\boldsymbol{\beta} + \boldsymbol{\epsilon}_{s_j}$ and $\mathbf{Y}_{-s_j} = \mathbf{X}_{-s_j}\boldsymbol{\beta} + \boldsymbol{\epsilon}_{-s_j}$. Hence, $E(\mathbf{Y}_{s_j}) = \mathbf{X}_{s_j}\boldsymbol{\beta}$ and $E(\hat{\mathbf{Y}}_{s_j}^{\text{cv}}) = \mathbf{X}_{s_j}E(\hat{\boldsymbol{\beta}}_{-s_j}) = \mathbf{X}_{s_j}\boldsymbol{\beta}$, so that $E(Y_{s_j,i} - \hat{Y}_{s_j,i}^{\text{cv}}) = 0$.

To calculate $\text{Var}(Y_{s_j,i} - \hat{Y}_{s_j,i}^{\text{cv}})$, we note that $\mathbf{Y}_{s_j}$ and $\hat{\mathbf{Y}}_{s_j}^{\text{cv}}$ are independent, so that

$$\text{Var}(Y_{s_j,i} - \hat{Y}_{s_j,i}^{\text{cv}}) = \text{Var}(Y_{s_j,i}) + \text{Var}(\hat{Y}_{s_j,i}^{\text{cv}}).$$

Now, $\text{Var}(Y_{s_j,i}) = \sigma^2$, and

$$\text{Var}(\hat{\mathbf{Y}}_{s_j}^{\text{cv}}) = \text{Var}(\mathbf{X}_{s_j}(\mathbf{X}_{-s_j}^T\mathbf{X}_{-s_j})^{-1}\mathbf{X}_{-s_j}^T\mathbf{Y}_{-s_j}) = \sigma^2\mathbf{X}_{s_j}(\mathbf{X}_{-s_j}^T\mathbf{X}_{-s_j})^{-1}\mathbf{X}_{s_j}^T.$$

Therefore, $\mathrm{Var}(\hat{Y}^{\mathrm{cv}}_{s_j,i}) = \sigma^2[\mathbf{X}_{s_j}(\mathbf{X}^T_{-s_j}\mathbf{X}_{-s_j})^{-1}\mathbf{X}^T_{s_j}]_{ii}$.

Hence we have

$$\begin{aligned} E(\mathrm{CVMSE}_{\mathbf{s}}) &= \frac{1}{n}\sum_{j=1}^{v}\sum_{i=1}^{n_{s_j}}\{\sigma^2 + \sigma^2[\mathbf{X}_{s_j}(\mathbf{X}^T_{-s_j}\mathbf{X}_{-s_j})^{-1}\mathbf{X}^T_{s_j}]_{ii}\} \\ &= \sigma^2 + \frac{\sigma^2}{n}\sum_{j=1}^{v}\mathrm{tr}[\mathbf{X}_{s_j}(\mathbf{X}^T_{-s_j}\mathbf{X}_{-s_j})^{-1}\mathbf{X}^T_{s_j}]. \qquad \text{(B.1)} \end{aligned}$$

By the matrix inverse updating formula (e.g., Searle, 1982, p. 261)

$$(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1} = \mathbf{D}^{-1} + \mathbf{D}^{-1}\mathbf{C}(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1}\mathbf{B}\mathbf{D}^{-1},$$

with $\mathbf{D} = \mathbf{X}^T\mathbf{X} = \mathbf{X}^T_{-s_j}\mathbf{X}_{-s_j} + \mathbf{X}^T_{s_j}\mathbf{X}_{s_j}$, $\mathbf{C} = \mathbf{X}^T_{s_j}$, $\mathbf{B} = \mathbf{X}_{s_j}$, $\mathbf{A} = \mathbf{I}$, we have

$$(\mathbf{X}^T_{-s_j}\mathbf{X}_{-s_j})^{-1} = (\mathbf{X}^T\mathbf{X} - \mathbf{X}^T_{s_j}\mathbf{X}_{s_j})^{-1} = (\mathbf{X}^T\mathbf{X})^{-1} + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T_{s_j}(\mathbf{I} - \mathbf{P}_{s_j})^{-1}\mathbf{X}_{s_j}(\mathbf{X}^T\mathbf{X})^{-1},$$

where $\mathbf{P}_{s_j}$ was given in (4.5). Pre- and post-multiplying both sides by $\mathbf{X}_{s_j}$ and $\mathbf{X}^T_{s_j}$ gives

$$\mathbf{X}_{s_j}(\mathbf{X}^T_{-s_j}\mathbf{X}_{-s_j})^{-1}\mathbf{X}^T_{s_j} = \mathbf{P}_{s_j} + \mathbf{P}_{s_j}(\mathbf{I} - \mathbf{P}_{s_j})^{-1}\mathbf{P}_{s_j}.$$

Therefore, in (B.1) we have

$$\sum_{j=1}^{v}\mathrm{tr}[\mathbf{X}_{s_j}(\mathbf{X}^T_{-s_j}\mathbf{X}_{-s_j})^{-1}\mathbf{X}^T_{s_j}] = \sum_{j=1}^{v}\mathrm{tr}(\mathbf{P}_{s_j}) + \sum_{j=1}^{v}\mathrm{tr}[\mathbf{P}_{s_j}(\mathbf{I} - \mathbf{P}_{s_j})^{-1}\mathbf{P}_{s_j}]. \qquad \text{(B.2)}$$

Noting that $\sum_{j=1}^{v}\mathrm{tr}(\mathbf{P}_{s_j}) = \mathrm{tr}(\mathbf{H}) = p$, where $\mathbf{H}$ was given in (4.7), we next calculate $\mathrm{tr}[\mathbf{P}_{s_j}(\mathbf{I} - \mathbf{P}_{s_j})^{-1}\mathbf{P}_{s_j}]$. Write the eigen decomposition of $\mathbf{P}_{s_j}$ as $\mathbf{U}_{s_j}\mathbf{D}_{s_j}\mathbf{U}^T_{s_j}$, where $\mathbf{U}_{s_j}$ is an orthogonal matrix, and $\mathbf{D}_{s_j}$ is the diagonal matrix with $d_{s_j,i}$ for $i = 1, \ldots, n_{s_j}$, i.e., the eigenvalues of $\mathbf{P}_{s_j}$, on the

diagonal. Hence,

$$(\mathbf{I}-\mathbf{P}_{s_j})^{-1}=\mathbf{U}_{s_j}(\mathbf{I}-\mathbf{D}_{s_j})^{-1}\mathbf{U}_{s_j}^T,$$

and

$$\mathbf{P}_{s_j}(\mathbf{I}-\mathbf{P}_{s_j})^{-1}\mathbf{P}_{s_j}=\mathbf{U}_{s_j}\mathbf{D}_{s_j}(\mathbf{I}-\mathbf{D}_{s_j})^{-1}\mathbf{D}_{s_j}\mathbf{U}_{s_j}^T,$$

whereupon

$$\mathrm{tr}[\mathbf{P}_{s_j}(\mathbf{I}-\mathbf{P}_{s_j})^{-1}\mathbf{P}_{s_j}]=\mathrm{tr}[\mathbf{D}_{s_j}^2(\mathbf{I}-\mathbf{D}_{s_j})^{-1}]=\sum_{i=1}^{n_{s_j}}\frac{d_{s_j,i}^2}{1-d_{s_j,i}}.$$

Combining this expression with (B.1) and (B.2) gives the result in Lemma 1.

B.1.2 Proof of Lemma 2

The proof follows the steps of Lemma 1. We again decompose $E(Y_{s_j,i}-\hat{Y}_{s_j,i}^{\mathrm{cv}})^2$ as $E^2(Y_{s_j,i}-\hat{Y}_{s_j,i}^{\mathrm{cv}})+\mathrm{Var}(Y_{s_j,i}-\hat{Y}_{s_j,i}^{\mathrm{cv}})$ and note that only $E(Y_{s_j,i}-\hat{Y}_{s_j,i}^{\mathrm{cv}})$ changes from Lemma 1.

As $\mathbf{Y}=\mathbf{X}^*\boldsymbol{\beta}^*+\boldsymbol{\epsilon}=\mathbf{X}\boldsymbol{\beta}+\mathbf{X}^u\boldsymbol{\beta}^u+\boldsymbol{\epsilon}$, we have

$$E(\mathbf{Y}_{s_j})=\mathbf{X}_{s_j}\boldsymbol{\beta}+\mathbf{X}_{s_j}^u\boldsymbol{\beta}^u$$

and

$$\begin{aligned}E(\hat{\mathbf{Y}}_{s_j}^{\mathrm{cv}})&=\mathbf{X}_{s_j}(\mathbf{X}_{-s_j}^T\mathbf{X}_{-s_j})^{-1}\mathbf{X}_{-s_j}^TE(\mathbf{Y}_{-s_j})=\mathbf{X}_{s_j}(\mathbf{X}_{-s_j}^T\mathbf{X}_{-s_j})^{-1}\mathbf{X}_{-s_j}^T(\mathbf{X}_{-s_j}\boldsymbol{\beta}+\mathbf{X}_{-s_j}^u\boldsymbol{\beta}^u)\\&=\mathbf{X}_{s_j}\boldsymbol{\beta}+\mathbf{X}_{s_j}(\mathbf{X}_{-s_j}^T\mathbf{X}_{-s_j})^{-1}\mathbf{X}_{-s_j}^T\mathbf{X}_{-s_j}^u\boldsymbol{\beta}^u.\end{aligned}$$

Therefore,

$$E(\mathbf{Y}_{s_j}-\hat{\mathbf{Y}}_{s_j}^{\mathrm{cv}})=\tilde{\mathbf{X}}_{s_j}^u\boldsymbol{\beta}^u,$$

where $\tilde{\mathbf{X}}^u_{s_j} = \mathbf{X}^u_{s_j} - \mathbf{X}_{s_j}(\mathbf{X}^T_{-s_j}\mathbf{X}_{-s_j})^{-1}\mathbf{X}^T_{-s_j}\mathbf{X}^u_{-s_j}$. Therefore,

$$\sum_{j=1}^{v}\sum_{i=1}^{n_{s_j}} E^2[Y_{s_j,i} - \hat{Y}^{\mathrm{cv}}_{s_j,i}] = \boldsymbol{\beta}^{u^T}\sum_{j=1}^{v}\tilde{\mathbf{X}}^{u^T}_{s_j}\tilde{\mathbf{X}}^u_{s_j}\boldsymbol{\beta}^u,$$

and this quantity must be added to the result in Lemma 1.

B.1.3 Proof of Lemma 3

From the proof of B.1.1, we know that

$$\sum_{i=1}^{n_{s_j}} \frac{d^2_{s_j,i}}{1-d_{s_j,i}} = \mathrm{tr}[\mathbf{P}_{s_j}(\mathbf{I}-\mathbf{P}_{s_j})^{-1}\mathbf{P}_{s_j}].$$

We will show that $(\mathbf{I}-\mathbf{P}_{s_j})^{-1}$ is positive definite. The result will then follow because $\mathbf{P}_{s_j}(\mathbf{I}-\mathbf{P}_{s_j})^{-1}\mathbf{P}_{s_j}$ is hence nonnegative definite and must have a nonnegative trace.

In the matrix inverse updating formula used for the proof of Lemma 1,

$$(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1} = \mathbf{D}^{-1} + \mathbf{D}^{-1}\mathbf{C}(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1}\mathbf{B}\mathbf{D}^{-1},$$

put $\mathbf{D}=\mathbf{I}$, $\mathbf{C}=\mathbf{X}_{s_j}$, $\mathbf{A}=\mathbf{X}^T\mathbf{X}$, and $\mathbf{B}=\mathbf{X}^T_{s_j}$. Then

$$\begin{aligned}(\mathbf{I}-\mathbf{P}_{s_j})^{-1} &= (\mathbf{I}-\mathbf{X}_{s_j}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T_{s_j})^{-1} = \mathbf{I}+\mathbf{X}_{s_j}(\mathbf{X}^T\mathbf{X}-\mathbf{X}^T_{s_j}\mathbf{X}_{s_j})^{-1}\mathbf{X}^T_{s_j})\\ &= \mathbf{I}+\mathbf{X}_{s_j}(\mathbf{X}^T_{-s_j}\mathbf{X}_{-s_j})^{-1}\mathbf{X}^T_{s_j},\end{aligned}$$

which is clearly positive definite.

B.1.4 Proof of Theorem 3

By assumption all the positive eigenvalues of $\mathbf{P}_{s_1}, \ldots, \mathbf{P}_{s_v}$ are the same, and hence equal to their arithmetic average, To compute the average, we note

$$\begin{aligned}\sum_{j=1}^{v}\sum_{i=1}^{n_{s_j}} d_{s_j,i} &= \sum_{j=1}^{v} \mathrm{tr}(\mathbf{P}_{s_j}) = \sum_{j=1}^{v} \mathrm{tr}(\mathbf{X}_{s_j}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}_{s_j}^T) \\ &= \sum_{j=1}^{v} \mathrm{tr}(\mathbf{X}_{s_j}\mathbf{X}_{s_j}^T) \quad \text{(because } \mathbf{X}^T\mathbf{X} = \mathbf{I} \text{ by assumption)} \\ &= \sum_{j=1}^{v} \mathrm{tr}(\mathbf{X}_{s_j}^T\mathbf{X}_{s_j}) = \mathrm{tr}\left(\sum_{j=1}^{v} \mathbf{X}_{s_j}^T\mathbf{X}_{s_j}\right) = \mathrm{tr}(\mathbf{X}^T\mathbf{X}) = \mathrm{tr}(\mathbf{I}) = p.\end{aligned}$$

Next we count the number of positive eigenvalues. As we are assuming each fold has the same number of observations, $n_{s_j} = n/v$ for $j = 1, \ldots, v$. Now we consider two cases:
(1) If $n/v > p$ (or $v < n/p$), the rank of $\mathbf{P}_{s_j}$ is p for $j = 1, \ldots, v$, the number of positive eigenvalues is vp, and the average of the positive eigenvalues is $p/(vp) = 1/v$.
(2) If $n/v \leq p$ (or $v \geq n/p$), the rank of $\mathbf{P}_{s_j}$ is n/v for $j = 1, \ldots, v$, the number of positive eigenvalues is $v(n/v) = n$), and the average of the positive eigenvalues is p/n.

The proof is completed by summing over only the positive $d_{s_j,i}$ in Theorem 1 and replacing these $d_{s_j,i}$ by the appropriate average. In case (1) above,

$$\frac{\sigma^2}{n}\sum_{j=1}^{v}\sum_{i=1}^{n_{s_j}} \frac{d_{s_j,i}^2}{1 - d_{s_j,i}} = \frac{\sigma^2}{n} vp \frac{1/v^2}{1 - 1/v} = \frac{p}{n(v-1)}\sigma^2,$$

whereas in case (2),

$$\frac{\sigma^2}{n}\sum_{j=1}^{v}\sum_{i=1}^{n_{s_j}} \frac{d_{s_j,i}^2}{1 - d_{s_j,i}} = \frac{\sigma^2}{n} n \frac{p^2/n^2}{1 - p/n} = \frac{p^2}{n(n-p)}\sigma^2.$$

B.1.5 Proof of Theorem 4

We have CVMSE $= \frac{1}{n}\mathbf{r}^T\mathbf{\Lambda}^2\mathbf{r}$. Hence,

$$\text{Var(CVMSE)} = \frac{1}{n^2}\text{Var}(\mathbf{r}^T\mathbf{\Lambda}^2\mathbf{r}).$$

Noting that $\mathbf{r} = (\mathbf{I} - \mathbf{H})\mathbf{Y}$, like $\mathbf{Y}$, is multivariate normal, we can a result on the variance of a quadratic form of normal random variables (e.g., Searle, 1982, p. 355),

$$\text{Var}(\mathbf{r}^T\mathbf{\Lambda}^2\mathbf{r}) = 2\text{tr}[\mathbf{\Lambda}^2\text{Cov}(\mathbf{r})]^2 + 4E(\mathbf{r}^T)\mathbf{\Lambda}^2\text{Cov}(\mathbf{r})\mathbf{\Lambda}^2E(\mathbf{r}). \tag{B.3}$$

To obtain the first term on the right-hand side of (B.3), we use

$$\text{Cov}(\mathbf{r}) = (\mathbf{I} - \mathbf{H})\text{Cov}(\mathbf{Y})(\mathbf{I} - \mathbf{H})^T = \sigma^2(\mathbf{I} - \mathbf{H}),$$

since $\mathbf{I} - \mathbf{H}$ is a projection matrix and $\text{Cov}(\mathbf{Y}) = \text{Cov}(\boldsymbol{\epsilon})$ is assumed to be $\sigma^2\mathbf{I}$. Thus, the first term is

$$2\text{tr}[\mathbf{\Lambda}^2\text{Cov}(\mathbf{r})]^2 = 2\sigma^4\text{tr}[\mathbf{\Lambda}^2(\mathbf{I} - \mathbf{H})]^2.$$

For the second term on the right-hand side of (B.3), we need $E(\mathbf{r}) = (\mathbf{I}-\mathbf{H})E(\mathbf{Y}) = (\mathbf{I}-\mathbf{H})\mathbf{X}^*\boldsymbol{\beta}^*$, which depends on whether there is overfitting or underfitting.

If we overfit, $(\mathbf{I} - \mathbf{H})\mathbf{X}^* = \mathbf{0}$, because $\mathbf{HX} = \mathbf{X}$ and $\mathbf{X}^*$ is a subset of the columns of $\mathbf{X}$. Hence $E(\mathbf{r}) = 0$ when overfitting, and there is no further contribution to (B.3).

If we underfit,

$$E(\mathbf{r}) = (\mathbf{I} - \mathbf{H})\mathbf{X}^*\boldsymbol{\beta}^* = (\mathbf{I} - \mathbf{H})(\mathbf{X}\boldsymbol{\beta} + \mathbf{X}^u\boldsymbol{\beta}^u) = (\mathbf{I} - \mathbf{H})\mathbf{X}^u\boldsymbol{\beta}^u,$$

again because $\mathbf{HX} = \mathbf{X}$. Thus, when underfitting, the second term on the

right-hand side of (B.3) is

$$4\sigma^2 \boldsymbol{\beta}^{u^T} \mathbf{X}^{u^T} (\mathbf{I}-\mathbf{H})\boldsymbol{\Lambda}^2(\mathbf{I}-\mathbf{H})\boldsymbol{\Lambda}^2(\mathbf{I}-\mathbf{H})\mathbf{X}^u\boldsymbol{\beta}^u.$$

B.1.6 Proof of Theorem 5

For the split $\mathbf{s}$, we have

$$\text{CVMSE}_{\mathbf{s}} = \frac{1}{n}\sum_{j=1}^{v} ||\mathbf{Y}_{s_j} - \hat{\mathbf{Y}}^{\text{cv}}_{s_j}||^2.$$

Herein, $\mathbf{Y}_{s_j} = \mathbf{X}^*_{s_j}\boldsymbol{\beta}^* + \boldsymbol{\epsilon}_{s_j}$ and

$$\begin{aligned}
\hat{\mathbf{Y}}^{\text{cv}}_{s_j} &= \mathbf{X}^*_{s_j}\hat{\beta}^*_{-s_j} = \mathbf{X}^*_{s_j}(\mathbf{X}^{*^T}_{-s_j}\mathbf{X}^*_{-s_j})^{-1}\mathbf{X}^{*^T}_{-s_j}\mathbf{Y}_{-s_j} \\
&= \frac{v}{v-1}\mathbf{X}^*_{s_j}(\mathbf{X}^{*^T}\mathbf{X}^*)^{-1}\mathbf{X}^{*^T}_{-s_j}(\mathbf{X}^*_{-s_j}\boldsymbol{\beta}^* + \boldsymbol{\epsilon}_{-s_j}) \\
&= \mathbf{X}^*_{s_j}\boldsymbol{\beta}^* + \frac{v}{v-1}\mathbf{X}^*_{s_j}(\mathbf{X}^{*^T}\mathbf{X}^*)^{-1}\mathbf{X}^{*^T}_{-s_j}\boldsymbol{\epsilon}_{-s_j},
\end{aligned}$$

where the last two equalities use $\mathbf{X}^{*^T}_{-s_j}\mathbf{X}^*_{-s_j} = \frac{v-1}{v}\mathbf{X}^{*^T}\mathbf{X}^*$, which follows from the condition $\mathbf{X}^{*^T}_{s_j}\mathbf{X}^*_{s_j} = \frac{1}{v}\mathbf{X}^{*^T}\mathbf{X}^*$ of the theorem. Hence,

$$\mathbf{Y}_{s_j} - \hat{\mathbf{Y}}^{\text{cv}}_{s_j} = \boldsymbol{\epsilon}_{s_j} - \frac{v}{v-1}\mathbf{X}^*_{s_j}(\mathbf{X}^{*^T}\mathbf{X}^*)^{-1}\mathbf{X}^{*^T}_{-s_j}\boldsymbol{\epsilon}_{-s_j},$$

and

$$\begin{aligned}
||\mathbf{Y}_{s_j} - \hat{\mathbf{Y}}^{\text{cv}}_{s_j}||^2 = \boldsymbol{\epsilon}^T_{s_j}\boldsymbol{\epsilon}_{s_j} &- \frac{2v}{v-1}\boldsymbol{\epsilon}^T_{s_j}\mathbf{X}^*_{s_j}(\mathbf{X}^{*^T}\mathbf{X}^*)^{-1}\mathbf{X}^{*^T}_{-s_j}\boldsymbol{\epsilon}_{-s_j} \\
&+ \frac{v}{(v-1)^2}\boldsymbol{\epsilon}^T_{-s_j}\mathbf{X}^*_{-s_j}(\mathbf{X}^{*^T}\mathbf{X}^*)^{-1}\mathbf{X}^{*^T}_{-s_j}\boldsymbol{\epsilon}_{-s_j}. \qquad \text{(B.4)}
\end{aligned}$$

The next step is to sum (B.4) over j. It is easily seen that the first term in (B.4) leads to $\sum_{j=1}^{v}\boldsymbol{\epsilon}^T_{s_j}\boldsymbol{\epsilon}_{s_j} = \boldsymbol{\epsilon}^T\boldsymbol{\epsilon}$. By writing $\mathbf{X}^*_{-s_j}\boldsymbol{\epsilon}_{-s_j} = \sum_{i\neq j}^{v}\mathbf{X}^*_{s_i}\boldsymbol{\epsilon}_{s_i}$,

the second term on the right side of (B.4) leads to

$$-\frac{2v}{v-1}\sum_{j=1}^{v}\sum_{i\neq j}^{v}\boldsymbol{\epsilon}_{s_j}^T\mathbf{X}_{s_j}^*(\mathbf{X}^{*^T}\mathbf{X}^*)^{-1}\mathbf{X}_{s_i}^{*^T}\boldsymbol{\epsilon}_{s_i}.$$

Similarly, the third term on the right side of (B.4) leads to

$$\frac{v}{(v-1)^2}\sum_{j=1}^{v}\sum_{i\neq j}^{v}\boldsymbol{\epsilon}_{s_i}^T\mathbf{X}_{s_i}^*(\mathbf{X}^{*^T}\mathbf{X}^*)^{-1}\mathbf{X}_{s_i}^{*^T}\boldsymbol{\epsilon}_{s_i}+\frac{v}{(v-1)^2}\sum_{j=1}^{v}\sum_{i\neq j}^{v}\sum_{k\neq i,j}^{v}\boldsymbol{\epsilon}_{s_i}^T\mathbf{X}_{s_i}^*(\mathbf{X}^{*^T}\mathbf{X}^*)^{-1}\mathbf{X}_{s_k}^{*^T}\boldsymbol{\epsilon}_{s_k}.$$

Combining these three contributions and counting terms leads to

$$\begin{aligned}\mathrm{CVMSE}_{\mathbf{s}} &= \frac{1}{n}\{\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}+\frac{v}{v-1}\sum_{j=1}^{v}\boldsymbol{\epsilon}_{s_j}^T\mathbf{X}_{s_j}^*(\mathbf{X}^{*^T}\mathbf{X}^*)^{-1}\mathbf{X}_{s_j}^{*^T}\boldsymbol{\epsilon}_{s_j}\\ &\quad-(\frac{v}{v-1})^2\sum_{j=1}^{v}\sum_{i\neq j}^{v}\boldsymbol{\epsilon}_{s_j}^T\mathbf{X}_{s_j}^*(\mathbf{X}^{*^T}\mathbf{X}^*)^{-1}\mathbf{X}_{s_i}^{*^T}\boldsymbol{\epsilon}_{s_i}\}\\ &= \frac{1}{n}\boldsymbol{\epsilon}^T\mathbf{Q}\boldsymbol{\epsilon},\end{aligned}$$

where $\mathbf{Q}$ is defined in Theorem 5.

Using a result for the variance of a quadratic form of normal random variables (e.g., Searle, 1982, p. 355), we get

$$\mathrm{Var}(\mathrm{CVMSE}_{\mathbf{s}}) = \frac{2\sigma^4}{n^2}\mathrm{tr}(\mathbf{Q}^2).$$

To compute $\mathrm{tr}(\mathbf{Q}^2)$, write $\mathbf{Q}$ as $(\mathbf{Q}_{ij})$, where

$$\mathbf{Q}_{ii} = \mathbf{I}+\frac{v}{v-1}\mathbf{X}_{s_i}^*(\mathbf{X}^{*^t}\mathbf{X}^*)^{-1}\mathbf{X}_{s_i}^{*^t}, i=1,\ldots,v,$$

$$\mathbf{Q}_{ij} = -(\frac{v}{v-1})^2\mathbf{X}_{s_i}^*(\mathbf{X}^{*^t}\mathbf{X}^*)^{-1}\mathbf{X}_{s_j}^{*^t}, i,j=1,\ldots,v.$$

Then $\mathrm{tr}(\mathbf{Q}^2) = \sum_{i=1}^{v}\mathrm{tr}(\mathbf{Q}_{ii}^2)+\sum_{i\neq j}\mathrm{tr}(\mathbf{Q}_{ij}\mathbf{Q}_{ji})$.

Under the assumption $\mathbf{X}_{s_i}^{*t}\mathbf{X}_{s_i}^{*} = \frac{1}{v}\mathbf{X}^{*t}\mathbf{X}^{*}$, $i = 1, \ldots v$, we have

$$\mathbf{Q}_{ii}^2 = \mathbf{I} + [\frac{2v}{v-1} + \frac{v}{(v-1)^2}]\mathbf{X}_{s_i}^{*}(\mathbf{X}^{*t}\mathbf{X}^{*})^{-1}\mathbf{X}_{s_i}^{*t},$$

and

$$\mathbf{Q}_{ij}\mathbf{Q}_{ji} = \frac{v^3}{(v-1)^4}\mathbf{X}_{s_i}^{*}(\mathbf{X}^{*t}\mathbf{X}^{*})^{-1}\mathbf{X}_{s_i}^{*t}.$$

Hence,

$$\begin{aligned}
\mathrm{tr}(\mathbf{Q}_{ii}^2) &= \mathrm{tr}(\mathbf{I}_{n_{s_i}\times n_{s_i}}) + [\frac{2v}{v-1} + \frac{v}{(v-1)^2}]\mathrm{tr}[\mathbf{X}_{s_i}^{*}(\mathbf{X}^{*t}\mathbf{X}^{*})^{-1}\mathbf{X}_{s_i}^{*t}] \\
&= n_{s_i} + [\frac{2v^2 - v}{(v-1)^2}]\frac{1}{v}\mathrm{tr}(\mathbf{I}_{p^*\times p^*}) \\
&= n_{s_i} + \frac{2v-1}{(v-1)^2}p^* \ ,
\end{aligned}$$

and we get

$$\sum_{i=1}^{v}\mathrm{tr}(\mathbf{Q}_{ii}^2) = n + \frac{(2v-1)v}{(v-1)^2}p^* \ .$$

Similarly,

$$\begin{aligned}
\mathrm{tr}(\mathbf{Q}_{ij}\mathbf{Q}_{ji}) &= \mathrm{tr}[\frac{v^3}{(v-1)^4}\mathbf{X}_{s_i}^{*}(\mathbf{X}^{*t}\mathbf{X}^{*})^{-1}\mathbf{X}_{s_i}^{*t}] \\
&= \frac{v^2}{(v-1)^4}p^* \ .
\end{aligned}$$

Therefore,

$$\sum_{i\neq j}\mathrm{tr}(\mathbf{Q}_{ij}\mathbf{Q}_{ji}) = v(v-1)\frac{v^2}{(v-1)^4}p^* = (\frac{v}{v-1})^3 p^*.$$

Thus we get

$$\mathrm{tr}(\mathbf{Q}^2) = n + \frac{(2v-1)v}{(v-1)^2}p^* + \frac{v^3}{(v-1)^3}p^* = n + \frac{3v^3 - 3v^2 + v}{(v-1)^3}p^*.$$